ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Generación de una librería RVC - CAL para la etapa de estimación de abundancias en el proceso de análisis de imágenes hiperespectrales

AUTOR: Raquel Lazcano López

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

TUTOR (o Director en su caso): Eduardo Juárez Martínez

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Juana María Gutiérrez Arriola

VOCAL: César Sanz Álvaro

SECRETARIO: Eduardo Juárez Martínez

Fecha de lectura:

Calificación:

El Secretario,

AGRADECIMIENTOS

El desarrollo de este proyecto pone fin a una de las etapas más importantes de mi vida, que probablemente no habría conseguido superar de la misma forma sin la ayuda y el apoyo de ciertas personas a lo largo de los cuatro años de carrera:

- Mi agradecimiento más sincero va para mi familia y, en especial, para mis padres y mi hermana, por haber demostrado día a día una fe y confianza absoluta en mí - incluso cuando a mí me faltaba - y, sobre todo, por sentirse siempre orgullosos de mí y de mis logros. Muchas gracias por apoyarme y animarme a seguir cuando parecía que tanto esfuerzo no era suficientemente compensado. ¡Os quiero!

- En segundo lugar, quiero agradecer a Daniel Madroñal, gran compañero y, sobre todo, mejor amigo, porque, sin su apoyo y su ayuda, terminar esta carrera -y, sobre todo, este proyecto - habría sido mucho más difícil. Pero, sobre todo, muchas gracias por tantos buenos ratos y por hacer que prácticamente vivir en la universidad haya sido algo divertido. ¡A por el siguiente reto!

- En tercer lugar, me gustaría agradecer a mis amigos, por ser un apoyo constante y por conseguir que me relaje y me olvide de todo por un rato. Marcos y Carlos, muchas gracias por estos cuatro años de carrera y por ayudarme siempre que lo he necesitado; María, muchas gracias por tu confianza y tus ánimos, y por haberte convertido en una gran amiga para mí. Y, sobre todo, muchísimas gracias a dos de las personas más importantes de mi vida: a Estefanía, por ser mi mejor amiga y, prácticamente, como una hermana para mí, porque su apoyo a lo largo de todos estos años sólo es equiparable al de mi propia familia; y, especialmente, a Alberto, por haberme hecho crecer y madurar, por ser mi apoyo fundamental, por su ayuda y, sobre todo, por su infinita comprensión y paciencia, aguantando lo que muchos otros no hubieran podido. ¡Muchas gracias!

- Por último, quisiera agradecer a mi tutor, Eduardo Juárez, por darme a conocer el campo de la investigación y, sobre todo, por proporcionarme la oportunidad de realizar este proyecto en el campo que más me ha gustado desde que empecé a estudiar - combinar telecomunicaciones y medicina -. Además, también me gustaría agradecer a Alejo Iván Arias la ayuda prestada para la realización de este trabajo, pero, especialmente, los buenos momentos en el centro de investigación. Y, para terminar, me gustaría agradecer a Juan Carlos y César, los maestros de laboratorio, por habernos ayudado tanto y habernos tratado tan bien. Ha sido un placer trabajar con vosotros. ¡Lo prometido es deuda!

Espero que la próxima etapa que voy a comenzar sea, como mínimo, tan enriquecedora como ésta y, sobre todo, que la compartáis conmigo. A todos vosotros, gracias por estar siempre ahí.

RESUMEN

Las imágenes hiperespectrales permiten extraer información con una gran resolución espectral, que se suele extender desde el espectro ultravioleta hasta el infrarrojo. Aunque esta tecnología fue aplicada inicialmente a la observación de la superficie terrestre, esta característica ha hecho que, en los últimos años, la aplicación de estas imágenes se haya expandido a otros campos, como la medicina y, en concreto, la detección del cáncer. Sin embargo, este nuevo ámbito de aplicación ha generado nuevas necesidades, como la del procesado de las imágenes en tiempo real. Debido, precisamente, a la gran resolución espectral, estas imágenes requieren una elevada capacidad computacional para ser procesadas, lo que imposibilita la consecución de este objetivo con las técnicas tradicionales de procesado. En este sentido, una de las principales líneas de investigación persigue el objetivo del tiempo real mediante la paralelización del procesamiento, dividiendo esta carga computacional en varios núcleos que trabajen simultáneamente.

A este respecto, en el presente documento se describe el desarrollo de una librería de procesado hiperespectral para el lenguaje RVC - CAL, que está específicamente pensado para el desarrollo de aplicaciones multimedia y proporciona las herramientas necesarias para paralelizar las aplicaciones. En concreto, en este Proyecto Fin de Grado se han desarrollado las funciones necesarias para implementar dos de las cuatro fases de la cadena de análisis de una imagen hiperespectral - en concreto, las fases de estimación del número de *endmembers* y de la estimación de la distribución de los mismos en la imagen -; conviene destacar que este trabajo se complementa con el realizado por Daniel Madroñal en su Proyecto Fin de Grado, donde desarrolla las funciones necesarias para completar las otras dos fases de la cadena.

El presente documento sigue la estructura clásica de un trabajo de investigación, exponiendo, en primer lugar, las motivaciones que han cimentado este Proyecto Fin de Grado y los objetivos que se esperan alcanzar con él. A continuación, se realiza un amplio análisis del estado del arte de las tecnologías necesarias para su desarrollo, explicando, por un lado, las imágenes hiperespectrales y, por otro, todos los recursos hardware y software necesarios para la implementación de la librería. De esta forma, se proporcionarán todos los conceptos técnicos necesarios para el correcto seguimiento de este documento. Tras ello, se detallará la metodología seguida para la generación de la mencionada librería, así como el proceso de implementación de una cadena completa de procesado de imágenes hiperespectrales que permita la evaluación tanto de la bondad de la librería como del tiempo necesario para analizar una imagen hiperespectral completa. Una vez expuesta la metodología utilizada, se analizarán en detalle los resultados obtenidos en las pruebas realizadas; en primer lugar, se explicarán los resultados individuales extraídos del análisis de las dos etapas implementadas y, posteriormente, se discutirán los arrojados por el análisis de la ejecución de la cadena completa, tanto en uno como en varios núcleos. Por último, como resultado de este estudio se extraen una serie de conclusiones, que engloban aspectos como bondad de resultados, tiempos de ejecución y consumo de recursos; asimismo, se proponen una serie de líneas futuras de actuación con las que se podría continuar y complementar la investigación desarrollada en este documento.

ABSTRACT

Hyperspectral imaging collects information from across the electromagnetic spectrum, covering a wide range of wavelengths. Although this technology was initially developed for remote sensing and earth observation, its multiple advantages - such as high spectral resolution - led to its application in other fields, as cancer detection. However, this new field has shown specific requirements; for example, it needs to accomplish strong time specifications, since all the potential applications - like surgical guidance or in vivo tumor detection - imply real-time requisites. Achieving this time requirements is a great challenge, as hyperspectral images generate extremely high volumes of data to process. For that reason, some new research lines are studying new processing techniques, and the most relevant ones are related to system parallelization: in order to reduce the computational load, this solution executes image analysis in several processors simultaneously; in that way, this computational load is divided among the different cores, and real-time specifications can be accomplished.

This document describes the construction of a new hyperspectral processing library for RVC - CAL language, which is specifically designed for multimedia applications and allows multithreading compilation and system parallelization. This Diploma Project develops the required library functions to implement two of the four stages of the hyperspectral imaging processing chain - *endmember* and abundance estimations -. The two other stages - dimensionality reduction and *endmember* extraction - are studied in the Diploma Project of Daniel Madroñal, which complements the research work described in this document.

The document follows the classical structure of a research work. Firstly, it introduces the motivations that have inspired this Diploma Project and the main objectives to achieve. After that, it thoroughly studies the state of the art of the technologies related to the development of the library. The state of the art contains all the concepts needed to understand the contents of this research work, like the definition and applications of hyperspectral imaging and the typical processing chain. Thirdly, it explains the methodology of the library implementation, as well as the construction of a complete processing chain in RVC - CAL applying the mentioned library. This chain will test both the correct behavior of the library and the time requirements for the complete analysis of one hyperspectral image, either executing the chain in one processor or in several ones. Afterwards, the collected results will be carefully analyzed: first of all, individual results -from *endmember* and abundance estimations stages - will be discussed and, after that, complete results will be studied; this results will be obtained from the complete processing chain, so they will analyze the effects of multithreading and system parallelization on the mentioned processing chain. Finally, as a result of this discussion, some conclusions will be gathered regarding some relevant aspects, such as algorithm behavior, execution times and processing performance. Likewise, this document will conclude with the proposal of some future research lines that could continue the research work described in this document.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ACRÓNIMOS

- **AMEE**: *Automatic Morphological Endmember Extraction*
- **CITSEM**: *Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad*
- **FCLS**: *Fully Constrained Least - Squares*
- **FET**: *Future & Emerging Technologies*
- **GDEM**: *Grupo de Diseño Electrónico y Microelectrónico*
- **HAnDy:** *Hyperspectral Algorithms Development*
- **HELICoiD**: *HypErspectraL Imaging Cancer Detection*
- **HSI**: *HyperSpectral Imaging*
- **HYSIME**: *Hyperspectral Signal Subspace Identification by Minimum Error*
- **IEA**: *Iterative Error Analysis*
- **ISRA**: *Image Space Reconstruction Algorithm*
- **ITK**: *Insight ToolKit (Insight Segmentation and Registration Toolkit)*
- **LSU**: *Linear Spectral Unmixing*
- **MNF**: *Minimum Noise Fraction*
- **NCLS**: *Non - negativity Constrained Least Squares*
- **N-FINDR**: *N – finder algorithm*
- **ORCC**: *Open RVC – CAL Compiler*
- **OSP**: *Orthogonal Subspace Projection*
- **OTB**: *ORFEO ToolBox*
- **PAPI**: *Performance Application Programming Interface*
- **PCA**: *Principal Component Analysis*
- **RVC-CAL**: *Reconfigurable Video Coding – CAL Actor Language*
- **SCLS**: *Sum-to-one Constrained Least Squares*
- **SNR**: *Relación señal a ruido (Signal to Noise Ratio)*
- **SPP**: *Spatial Pre - Processing*
- **SSEE**: *Spatial – Spectral Endmember Extraction*
- **UCLS**: *Unconstrained Least Squares*
- **UPM**: *Universidad Politécnica de Madrid*
- **VCA**: *Vertex Component Analysis*
- **VD**: *Virtual Dimensionality*
- **VNL**: *Vision Numeric Library*

CAPÍTULO 1. INTRODUCCIÓN

1.1. Motivaciones

El presente trabajo se inscribe dentro de HELICoiD[1] (en inglés, *HypErspectraL Imaging Cancer Detection*), que es un proyecto europeo enmarcado dentro del Séptimo Programa Marco de la Unión Europea y, en concreto, en el FET – Open (en inglés, *Future & Emerging Technologies*). El proyecto está compuesto por miembros de varios países, como España, Francia, Reino Unido y Holanda. De entre todos ellos, este trabajo de investigación se desarrolla en el seno del GDEM (*Grupo de Diseño Electrónico y Microelectrónico*), perteneciente al CITSEM (*Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad*) de la UPM (*Universidad Politécnica de Madrid*).

En concreto, el objetivo de HELICoiD es tratar de generalizar una metodología para discriminar entre tejido sano y canceroso durante intervenciones quirúrgicas en tiempo real, empleando para ello técnicas de procesado de imágenes hiperespectrales, ya que, gracias a ellas, se puede extraer la firma espectral de cada tejido. Como el cáncer lleva asociado un cambio en la fisiología celular, esto debería detectarse como un cambio en la firma espectral; basándose en esta premisa, HELICoiD tratará de determinar si es posible extraer la firma espectral de cada célula cancerígena y, así, generalizar un modelo para la localización de las mismas.

Como socio del proyecto, el GDEM es líder del paquete de trabajo relacionado con la implementación de los algoritmos desarrollados en una plataforma multinúcleo -así como la elección de la misma - y en el posterior análisis de su funcionamiento para detectar las tareas críticas que ralentizan el procesamiento y, en consecuencia, alejan el objetivo del tiempo real. Por tanto, uno de los grandes retos de este proyecto es la consecución del análisis de las imágenes hiperespectrales en tiempo real. Para ello, primero se ha de explicar qué se entiende por tiempo real en el ámbito de HELICoiD. Si bien, de forma genérica, el tiempo real se establece en un rango de entre 30 y 40 cuadros por segundo, las características del proyecto hacen que se pueda reducir esa definición. Esto se debe a que, durante una cirugía, las imágenes captadas están centradas en un mismo plano que únicamente cambia con las intervenciones del cirujano; éstas serán precisas e introducirán variaciones mínimas y lentas, lo que hace que la definición clásica de tiempo real se pueda reducir para este ámbito.

Una vez enmarcado y contextualizado el proyecto HELICoiD y, en concreto, el trabajo a desarrollar por el GDEM, se va a describir brevemente el objeto de esta investigación. En primer lugar, se estudiarán los algoritmos existentes para el procesado de imágenes hiperespectrales para, en segundo lugar, elaborar una librería que reúna las funciones necesarias para poder implementar los mismos y que, además, proporcione recursos para paralelizar el sistema, así como para evaluar el rendimiento y la velocidad del sistema.

Dicha librería va a ser implementada en un lenguaje de flujo de datos como el lenguaje de programación de alto nivel RVC-CAL. Este lenguaje está basado en una serie de componentes modulares – o actores – que intercambian flujos de datos. Estos actores son independientes entre sí y pueden ejecutarse de forma simultánea. Por ello, un sistema modelado en RVC-CAL puede interpretarse como un grafo en el que los nodos son actores y los arcos, palabras. Esta nueva forma de entender un sistema permite cambiar los flujos entre actores, obteniendo una nueva configuración del sistema; y, además, controlar la división de la carga de trabajo en varios procesadores, ya que cada actor puede configurarse en un hilo y asignarse a un procesador.

[1] http://helicoid.eu/

Además, para medir el rendimiento del sistema se va a emplear la herramienta PAPI (*Performance Application Programming Interface*), que ha sido utilizada, con resultados satisfactorios, en la monitorización de sistemas para la reducción del consumo de energía, que es otra de las líneas de investigación del GDEM. Aplicándola a los algoritmos de detección de tejidos tumorales, se podrá analizar cuántos recursos consume cada actor y, tras ello, investigar de forma eficiente cómo mejorar dicho consumo para reducir el tiempo de ejecución. Cabe destacar que, en esta investigación, se ha utilizado una aplicación desarrollada por el compañero Alejo Iván Arias para utilizar PAPI de una forma sencilla y rápida.

En consecuencia, se puede afirmar que la implementación de la librería en este lenguaje no sólo es altamente recomendable, sino casi estrictamente necesaria, puesto que facilita los medios necesarios para paralelizar el sistema e integrar una herramienta como PAPI. Por tanto, el desarrollo de una librería que comprenda todas las funciones necesarias para la implementación de los algoritmos de procesado de imágenes hiperespectrales constituirá una herramienta básica y esencial para la consecución de los objetivos planteados en el paquete de trabajo asignado al grupo. Además de posibilitar la paralelización del sistema, esta librería aportará las rutinas necesarias para simplificar la tarea del programador a la hora de implementar estos algoritmos, proporcionando un nuevo nivel de abstracción en RVC - CAL.

1.2. Objetivos

Como se ha mencionado en el apartado anterior, el principal objetivo de este trabajo es el desarrollo una librería para la implementación de algoritmos de procesado de imágenes hiperespectrales en RVC - CAL. La integración de una herramienta que permita medir el rendimiento del sistema completa los objetivos generales de este Proyecto Fin de Grado. Para la consecución de este objetivo global, se han llevado a cabo los siguientes objetivos específicos:

- ✓ Adquirir conocimientos sobre las imágenes hiperespectrales, sus fases de procesado, sus aplicaciones y la metodología existente para aproximarse al tiempo real.
- ✓ Implementar la librería como una librería externa de RVC - CAL que permita combinar el potencial de un lenguaje tradicional, como C, con las ventajas que ofrece RVC - CAL con respecto al paralelismo, que ya han sido previamente mencionadas.
- ✓ Implementar una cadena completa de procesado de imágenes hiperespectrales para construir la librería, probar su correcto funcionamiento y, además, demostrar la simplificación en lo referente a nivel de abstracción y tiempo de programación.
- ✓ Desarrollar la interfaz de entrada y de salida del sistema, así como la interfaz entre fases de la cadena, con el fin de extraer el paralelismo y realizar una división eficiente entre procesadores para poder aproximarse, en la medida de lo posible, al tiempo real.
- ✓ Analizar de forma exhaustiva tanto la bondad de los algoritmos desarrollados como el tiempo de ejecución de los mismos, tanto por separado como dentro de la cadena de procesado, comprobando así el funcionamiento de la librería implementada.
- ✓ Aplicar la herramienta PAPI a la cadena de procesado para estudiar el consumo de recursos y localizar los puntos críticos de la misma, extrayendo así las posibles distribuciones en procesadores y midiendo la eficiencia de cada una de ellas para tratar de alcanzar el objetivo de tiempo real.

Conviene destacar que este Proyecto Fin de Grado y el desarrollado por Daniel Madroñal Quintín [1] comparten objetivos, puesto que se han desarrollado en paralelo y en la misma línea de investigación. Por tanto, su trabajo será referenciado con asiduidad en este documento.

CAPÍTULO 2. ANTECEDENTES

En este capítulo se van a introducir los principales conceptos relacionados con este Proyecto Fin de Grado, que estarán presentes a lo largo de todo el documento. De todos ellos se proporcionará una definición formal y se estudiará su estado del arte, analizando los campos de aplicación presentes y futuros.

En primer lugar, se definirá el concepto de imagen hiperespectral, resaltando las características principales de este tipo de imágenes. Además, se proporcionará información sobre el proceso de análisis de dichas imágenes, describiendo las distintas fases que componen la cadena de procesado de las mismas y explicando brevemente las distintas metodologías empleadas para ello.

En segundo lugar, se realizará un amplio análisis del estado del arte en lo referente a los ámbitos de aplicación de las imágenes hiperespectrales. En este apartado se hará especial hincapié en la aplicación de esta tecnología al mundo de la medicina y, en concreto, a la detección y localización de distintos tipos de tumores cancerígenos. Como se mencionó en el capítulo anterior, esto se debe a que este Proyecto se inscribe dentro del marco de HELICoiD, que es un proyecto europeo de aplicación de imágenes hiperespectrales a la detección del cáncer.

En tercer lugar, se realizará un estudio del estado del arte en lo concerniente a las velocidades de procesamiento de imágenes hiperespectrales, puesto que, como se mencionó en el capítulo 1, uno de los principales objetivos de este Proyecto Fin de Grado es acercarse, en la medida de lo posible, a la consecución del tiempo real en el análisis de este tipo de imágenes.

A continuación, se realizará un análisis del estado del arte en lo referente a las distintas plataformas multinúcleo que existen en el mercado en la actualidad, analizando brevemente sus prestaciones. El objetivo de este estudio es esbozar la elección de una plataforma sobre la que implementar la librería desarrollada, puesto que, aunque esta primera versión sólo se implementa en un ordenador, uno de los objetivos finales a medio plazo es la implementación de la misma en una plataforma multinúcleo.

Posteriormente, se introducirá el lenguaje RVC – CAL, ya que, como también mencionamos en el capítulo 1, otro de los objetivos principales de este Proyecto Fin de Grado radica en la implementación de la cadena de procesado de imágenes hiperespectrales en este nuevo lenguaje. En concreto, se definirá el lenguaje y se explicarán sus principales ventajas e inconvenientes, resaltando las razones que han motivado la decisión de utilizar este lenguaje para la implementación del proyecto.

Por último, para concluir este capítulo, se realizará una introducción de la librería PAPI (*Performance Application Programming Interface* o Interfaz de Programación para el Rendimiento de Aplicaciones), poniendo de manifiesto sus características principales. Asimismo, se analizarán los ámbitos de aplicación de la misma, resaltando, de forma concreta, el objetivo de su inclusión en nuestro proyecto, así como los resultados que se esperan conseguir.

Se ha de resaltar que este estudio se ha desarrollado en conjunto con Daniel Madroñal Quintín, por lo que su trabajo será referenciado con asiduidad a lo largo de este capítulo [1].

2.1. Imágenes hiperespectrales: definición y cadena de procesado

La evolución de las tecnologías de teledetección - del inglés *remote sensing* - para la observación remota de la superficie terrestre durante los últimos años del pasado siglo dio lugar a un nuevo campo de investigación, denominado espectroscopia de imagen - o, en inglés, *hyperspectral imaging*- [2, 24]. El objetivo de este nuevo campo es la obtención, estudio e interpretación de la radiación electromagnética procedente de la interacción entre una fuente de radiación, como puede ser un foco de luz, y el objeto bajo estudio. La base de la espectroscopia se apoya en que todos los materiales interaccionan con las ondas electromagnéticas, ya sea absorbiendo, reflejando o emitiendo energía electromagnética, y que, además, esta energía presenta unos patrones determinados y específicos de la composición molecular de cada material, además de estar localizados en distintas longitudes de onda [3]. Esto hace que de cada uno de ellos se pueda extraer una curva espectral única y característica, denominada *firma espectral*, que sirva para identificarlos de forma inequívoca. Esta curva puede obtenerse mediante sensores que miden, en las longitudes de onda de interés, parámetros como la *reflectancia* - o porcentaje de luz reflejada por un material o superficie.

Esta radiación electromagnética no sólo se limita al espectro visible, sino que abarca la mayor parte del espectro electromagnético, que se define como la distribución energética del conjunto de radiaciones electromagnéticas. La figura 2.1 muestra un claro ejemplo de este concepto: como se puede observar, la firma espectral - obtenida mediante la medida de la reflectancia - de tres materiales distintos, como son el agua, el suelo y la vegetación, se extiende mucho más allá del espectro visible, llegando a presentar picos en la zona del infrarrojo medio.

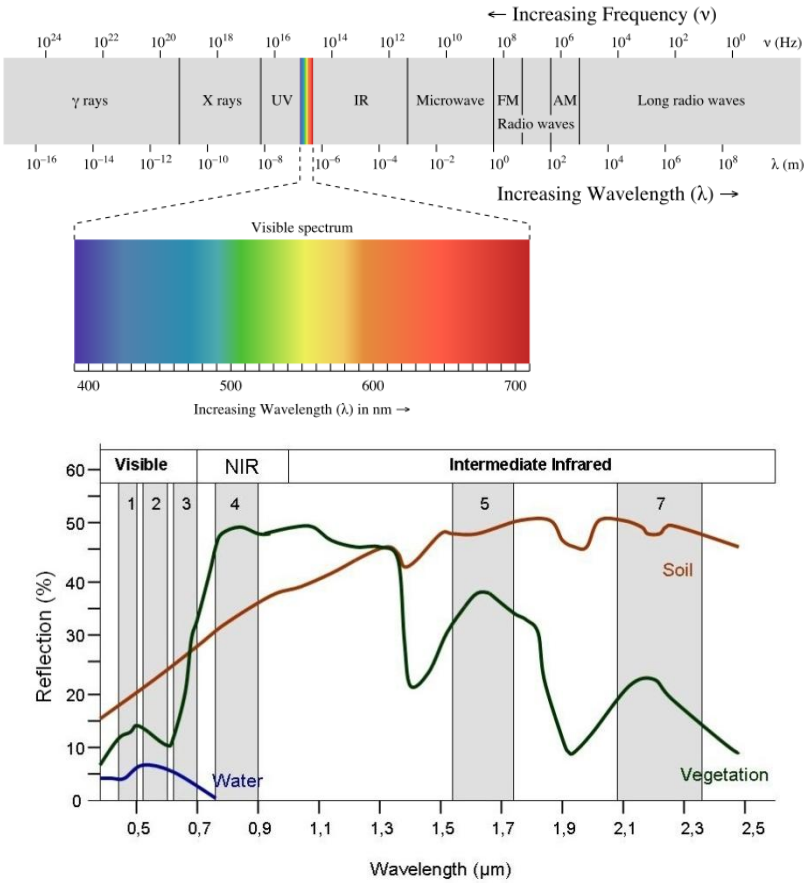


Fig. 2.1. Espectro electromagnético (arriba); firmas espectrales de suelo, agua y vegetación (abajo) (Fuente: www.seos-project.eu)

La extracción de las firmas espectrales de los materiales observados de forma remota puso de manifiesto la necesidad de desarrollar sensores que fueran capaces de captar imágenes no sólo dentro del espectro visible, sino también más allá de él. Esta necesidad tuvo como resultado la ampliación del concepto de píxel, dando lugar a la concepción de las llamadas *imágenes multidimensionales*. Se ha de recordar que, en una imagen puramente espacial - esto es, sin componente espectral -, el píxel lleva asociado un único valor discreto, mientras que, en las imágenes con componente espectral, el píxel está compuesto por un conjunto de valores.

En consecuencia, una imagen multidimensional se puede definir como aquella imagen en la que cada píxel no está exclusivamente descrito por un único valor de intensidad – como es el caso de las imágenes en blanco y negro – o por tres componentes de color – como en las imágenes RGB –, sino por un vector N-dimensional, donde N se corresponde con el número de bandas espectrales, en el que cada uno de sus valores se corresponde con la contribución de la luz detectada en ese punto para una banda concreta del espectro electromagnético [4]. El número de bandas para el que se obtienen estos valores espectrales – o lo que es lo mismo, el tamaño de este vector o N – puede extenderse desde unas cuantas decenas hasta varios centenares, comprendiendo, en la mayoría de los casos, el espectro infrarrojo y ultravioleta, y no únicamente el visible. Atendiendo a la variación de N, se puede extraer una clasificación de los distintos tipos de imágenes multidimensionales:

- Si el valor de N es reducido – típicamente, de unas decenas de bandas –, habitualmente se denominan imágenes multiespectrales.
- Si el valor de N aumenta hasta varias centenas de bandas, se habla de imágenes hiperespectrales.
- Si el valor de N crece hasta miles de bandas, reciben la denominación de imágenes ultraespectrales.

Además del número de bandas, las imágenes multiespectrales y las hiperespectrales presentan otra diferencia sustancial. En la figura 2.2 puede observarse que las bandas utilizadas en imágenes multiespectrales no son necesariamente contiguas, mientras que las bandas que componen las imágenes hiperespectrales sí han de serlo. Esto produce un efecto directo respecto a los resultados que se pueden obtener con cada una de ellas: con una imagen multidimensional se pueden obtener intensidades luminosas para bandas discretas, mientras que la imagen hiperespectral proporciona un espectro continuo del material bajo estudio.

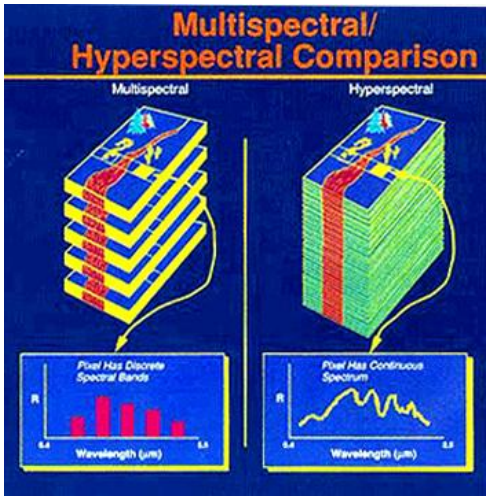


Fig. 2.2. Diferencia entre imágenes multiespectrales e hiperespectrales
(Fuente: http://landsat.gsfc.nasa.gov/)

Estas diferencias, y otras no mencionadas, se recogen en la figura 2.3, que proporciona una clasificación formal de las imágenes multidimensionales, atendiendo a parámetros como número de bandas, resolución, aplicaciones y disponibilidad.

DEFINITIONS OF IMAGING MODES

Mode	Number of Spectral Bands	Spectral Resolution	Capability	Availability
Imaging	None	None, sensitivity depends on detector spectral response	Image brightness	Now
Multispectral	Few to tens	Medium, many tens of nm	Detects solids and liquids	Now
Hyperspectral	Hundreds to ~thousands	Narrow, few nm	Detects and identifies solids and liquids	Now
Ultraspectral	Thousands	Very narrow	Detects and identifies solids, liquids and gases	Emerging technology but very expensive and processor hungry
Full spectrum	Thousands to "continuous spectra" over full optical spectral range from UV to IR	Very narrow	Detects and identifies solids, liquids and gases	Proposed technology and data processing system

Fig. 2.3. Clasificación de las imágenes multidimensionales (Fuente: Headwall Photonics)

En consecuencia, el conjunto de M vectores N-dimensionales – donde M es el número de píxeles de la imagen – da como resultado una matriz de tamaño variable, en la que sus dimensiones vendrán dadas por M y N. La representación de este tipo de imágenes suele ser el llamado *cubo de datos* o *hipercubo*, que emplea las dos dimensiones habituales para la representación de la ubicación espacial de cada píxel, y una tercera dimensión para representar la información espectral de cada píxel en las diferentes bandas [5].

En la figura 2.4. (a) se observa un ejemplo de una imagen hiperespectral real representada como un cubo de datos. En ella se pueden observar las dos dimensiones espaciales – denotadas x e y – y la tercera dimensión, espectral – denotada z –, de tal forma que, para cada una de las bandas que componen dicha imagen, se obtendría un mapa de luminosidad de la imagen captada – de forma análoga a una imagen en blanco y negro –. A la vista del cubo de datos mostrado en la figura 2.4, se podría decir, de una forma menos formal, que una imagen hiperespectral está compuesta de tantas imágenes en blanco y negro como bandas espectrales presenta dicha imagen - podría equipararse, en volumen, con N imágenes en blanco y negro.

Una de las principales ventajas de este tipo de representación es que permite la rápida extracción de la ya mencionada firma espectral de un determinado píxel. Para este tipo de imágenes, la firma espectral es el conjunto de valores de luminosidad - o reflectancia – que presenta un determinado píxel para cada una de las bandas que componen la imagen hiperespectral. En el cubo de datos, la firma espectral de un píxel (x', y') se puede obtener, tal y como observamos en la figura 2.4. (b), extrayendo de la imagen hiperespectral el vector (x', y',

z). De esta forma, si, como aparece en la figura 2.4. (b), se representan en un eje de coordenadas los valores de ese vector, se obtendrá la curva que conforma la firma hiperespectral de ese píxel.

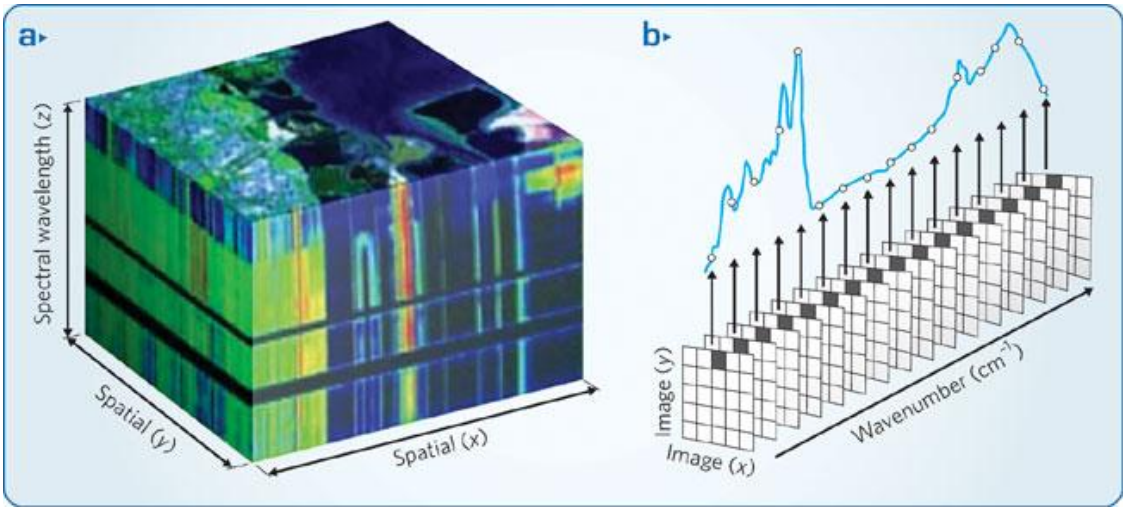


Fig. 2.4. (a) Ejemplo de una imagen hiperespectral real representada con un cubo de datos. (b) Obtención de la firma espectral de un determinado píxel (Fuente: NASA/Headwall Photonics)

Esta firma espectral característica de cada píxel será utilizada durante el proceso de análisis para determinar qué materiales se encuentran en cada píxel de la imagen. De esta última frase, puede deducirse que, en general, un píxel no estará conformado por un único material, sino que pueden aparecer varios en uno de ellos. Esto es fácilmente comprensible en aplicaciones como la observación de la superficie terrestre. Para aportar un ejemplo significativo, se puede destacar que el sensor *Airbone Visible/Infrared Imaging Spectometer* (AVIRIS[2]) desarrollado por la NASA (de sus siglas en inglés, *National Aeronautics and Space Administration*) presenta una resolución espacial de 20 metros por píxel [6]. Este dato ilustra de forma precisa la gran diversidad de materiales que puede existir en un único píxel, puesto que, por ejemplo, un píxel de una zona habitada podría estar compuesto por materiales como vegetación, edificios, calzadas, etc.

El ejemplo anterior sirve para introducir la clasificación general de los tipos de píxel que se pueden encontrar en el campo de las imágenes hiperespectrales. Esta clasificación consta, básicamente, de dos tipos de píxeles: los llamados píxeles *puros* y píxeles *mezcla* [3, 7]. La definición de estas categorías es intuitiva: los píxeles puros son aquellos en los que sólo aparece un único material y, en consecuencia, la firma espectral del píxel coincide con la de dicho material; por su parte, un píxel mezcla es aquél en el que aparecen varios materiales y, por tanto, no es posible, a priori, asignar la firma de ese píxel a un material concreto. La figura 2.5 muestra un ejemplo de ambos tipos de píxeles; como se puede observar, el píxel del agua se considera píxel puro y, en consecuencia, su firma espectral se corresponde con la del material, mientras que los píxeles que contienen vegetación y rocas son mezcla, puesto que, además de esos materiales, también aparece el material suelo. Cabe destacar que los píxeles mezcla pueden clasificarse, a su vez, en función del tipo de mezcla que presentan: *macroscópica* o *íntima* [8]. La mezcla macroscópica se produce cuando el tamaño del píxel imposibilita la separación de diferentes materiales, mientras que, por otro lado, la mezcla íntima se presenta cuando diferentes materiales se combinan íntimamente y, a simple vista, no se pueden discernir. Esto se

[2] http://aviris.jpl.nasa.gov

puede observar en la figura 2.6, donde se observa un píxel con una mezcla de cada tipo. Como se puede observar, la mezcla macroscópica está compuesta de un 15% de suelo, un 25% de árbol y un 60% de pasto, mientras que la mezcla íntima está compuesta de roca y arena y no se puede distinguir uno de otro.

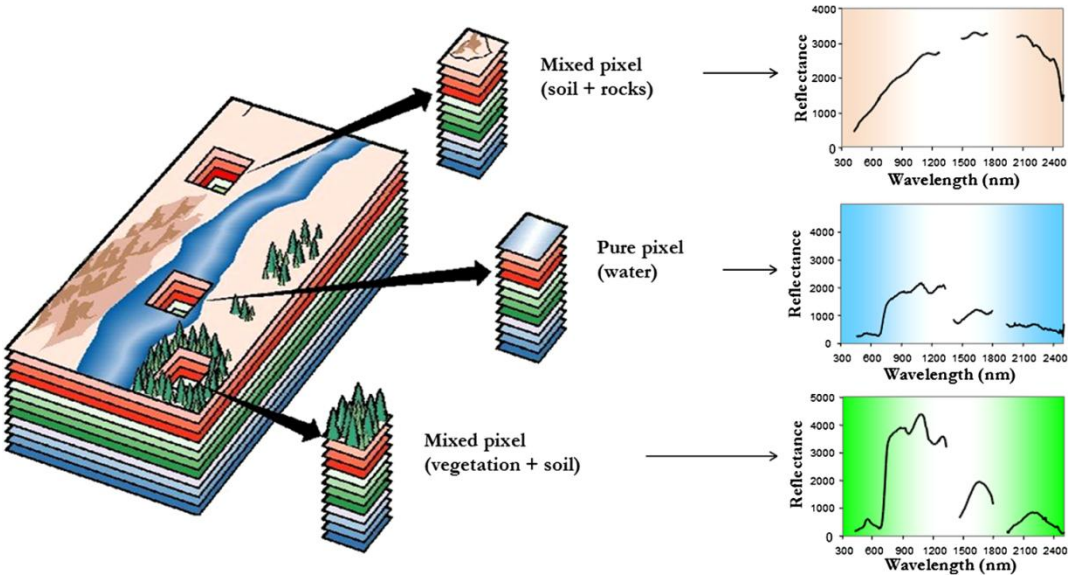


Fig. 2.5. Tipos de píxeles en las imágenes hiperespectrales (Fuente: SPIE Digital Library)

A la vista de todo lo anterior, parece evidente que la existencia de los píxeles mezcla genera una problemática:

- En primer lugar, se necesita descomponer la firma espectral de dicho píxel de tal forma que, de ella, se pueda extraer la firma de cada uno de los materiales presentes en ese píxel. Dicho de otro modo, este primer paso pretende descomponer la firma del píxel como una combinación de las firmas espectrales de los materiales existentes. La firma característica de un material presente en un píxel recibe el nombre de *endmember*; en consecuencia, se puede afirmar que los *endmembers* representan a los materiales espectralmente *puros* que aparecen en el píxel [2, 7, 9, 10].
- Posteriormente a este paso, la siguiente información que se necesita extraer es en qué porcentaje o proporción aparece cada uno de estos *endmembers* en dicho píxel. Esta proporción recibe el nombre de *fracciones de abundancia* - o, simplemente, *abundancia* -, y representa la distribución espacial de cada *endmember* en los distintos píxeles que conforman la imagen [2, 7, 9, 10].

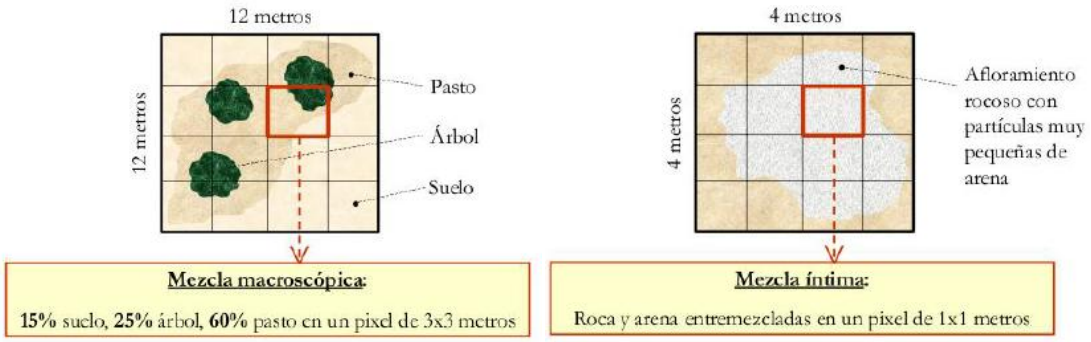


Fig. 2.6. Mezcla macroscópica (izquierda) y mezcla íntima (derecha)
(Fuente: http://www.umbc.edu/)

La problemática anterior ha sido profundamente analizada en la literatura referente al procesado de imágenes hiperespectrales, y ha recibido el nombre de *desmezclado espectral* o *spectral unmixing*. En concreto, según Keshava [8, 10], el desmezclado espectral es "el procedimiento por el cual el espectro medido de un píxel mezcla es descompuesto en una colección de espectros constituyentes, o *endmembers*, y en un conjunto de las correspondientes fracciones - o *abundancias* - que indican la proporción de cada *endmember* presente en dicho píxel".

A lo largo de los últimos años se han planteado multitud de soluciones al problema de la mezcla, pero todas convergen en la necesidad de modelar este comportamiento. En consecuencia, se han establecido dos modelos de mezcla, que se pueden caracterizar como modelo lineal y modelo no lineal.

El modelo lineal está pensado, principalmente, para escalas macroscópicas de mezcla; en este sentido, dicho modelo supone que los haces de radiación inciden únicamente con un solo *endmember*, de tal forma que la reflectancia total medida para un píxel mezcla puede expresarse como una combinación lineal de las firmas espectrales puras que aparecen en dicho píxel, junto con sus correspondientes abundancias [8, 10]. En la figura 2.7 se proporciona un ejemplo de este modelo: a la izquierda, se observa el proceso de adquisición de una imagen hiperespectral, en la que existen tres *endmembers* - denominados m_1, m_2 y m_3-; a la derecha, se aprecia la firma espectral extraída por el sensor. Como se puede observar, dicha firma está compuesta por una combinación lineal de las firmas espectrales de los distintos *endmembers* presentes en la imagen, donde la cantidad relativa de cada material viene dada por, en este caso, tres escalares - α_1, α_2 y α_3 - correspondientes a las abundancias características de cada *endmember* [11, 12].

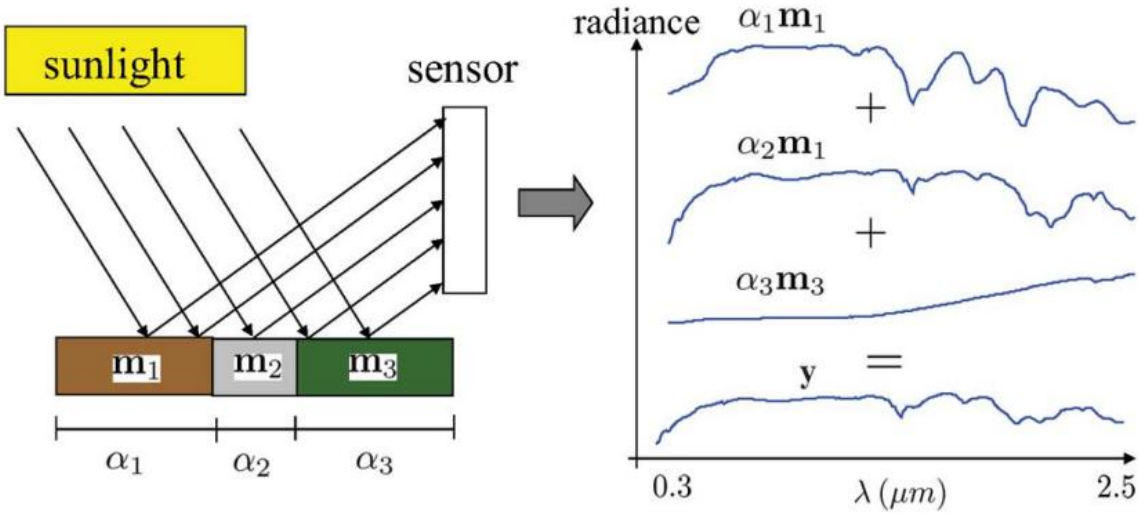


Fig. 2.7. Modelo lineal de mezcla (Fuente: http://www.umbc.edu/)

La formalización matemática de este modelo lineal sería la siguiente:

Dada una imagen hiperespectral con N bandas y M endmembers, si se denota s_i *a la firma espectral del i-ésimo endmember,* a_i *a su correspondiente abundancia y* w *al factor de ruido [10], se obtiene que el espectro* x *observado para cualquier píxel puede ser expresado como:*

$$x = \sum_{i=1}^{M} a_i \cdot s_i + w \qquad (2.1)$$

Por su parte, el modelo de mezcla no lineal supone un escenario más complicado, en el que la mezcla a nivel sub-píxel es íntima y, en consecuencia, no se puede asumir que la firma espectral total sea una combinación lineal de los *endmembers* y las abundancias, como asumía el modelo lineal. El modelo no lineal supone que los materiales no siguen una distribución superficial homogénea, sino que están distribuidos de forma aleatoria [10]. Como consecuencia de ello, la radiación incidente no interacciona únicamente con un solo endmember, sino que sufre múltiples reflexiones con varios materiales y, por tanto, el espectro resultante no puede medirse bajo parámetros lineales.

La figura 2.8 proporciona una comparativa visual de ambos modelos, en la que puede observarse cómo varía la reflexión de la luz en cada uno de ellos. De ella puede deducirse fácilmente que la extracción de un desarrollo matemático que modele los efectos de la mezcla no lineal conllevará una mayor complejidad [8, 10].

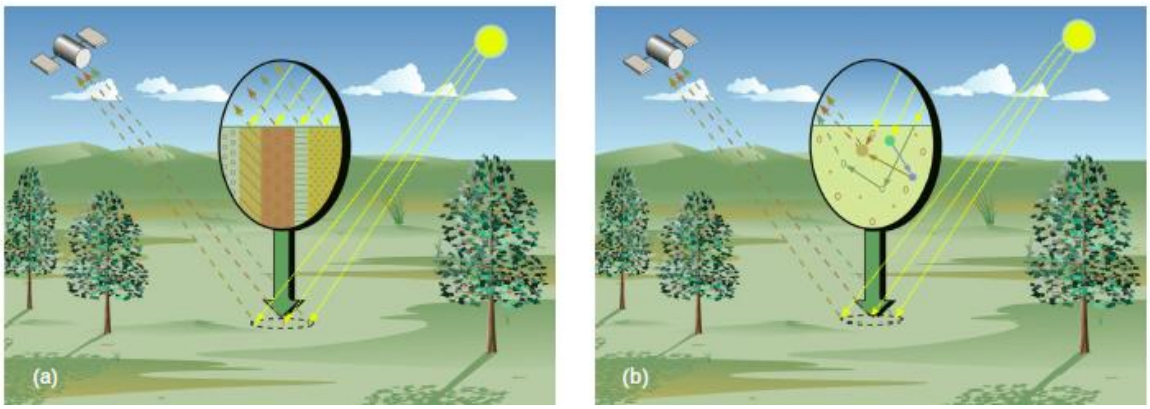


Fig. 2.8. Comparativa de los modelos de mezcla: (a) Modelo lineal (b) Modelo no lineal (Fuente: IEEE)

Teniendo en cuenta esta considerable dificultad, la tendencia de los últimos años a la hora de elegir un modelo u otro ha sido asumir que, dentro de lo posible, el modelo lineal proporciona una aproximación aceptable para resolver el problema de la mezcla. Si bien se puede afirmar que el modelo no lineal supone una aproximación mucho más acertada y robusta del comportamiento real de la luz, las múltiples ventajas del modelo lineal - como, por ejemplo, su generalidad y sencillez - han hecho que éste sea el modelo elegido para el desarrollo de este Proyecto Fin de Grado. Cabe mencionar que otro motivo que ha desencadenado esta decisión es que, actualmente, los desarrollos no lineales requieren una gran capacidad de procesamiento y, además, un tiempo de ejecución muy elevado, lo que imposibilitaría la consecución de uno de los objetivos más importantes - el tiempo real -. Consecuentemente, en lo sucesivo este Proyecto se centrará en el mencionado modelo lineal para el desmezclado espectral.

Por tanto, una vez que se ha determinado el modelo de mezcla a utilizar, se va a abordar el proceso de análisis de la imagen hiperespectral, explicando brevemente cada una de las etapas que componen una cadena completa de desmezclado espectral. Esta cadena parte, a su entrada, de la imagen original X captada por el sensor, y proporciona dos resultados: por un lado, un conjunto de m firmas espectrales o *endmembers* y, por otro, otro conjunto con las correspondientes abundancias de cada píxel.

De forma general, la forma de representar el resultado de estas abundancias es mediante un mapa de bits, por lo que también se podrá referir este último conjunto como *mapa de abundancia*, de tal forma que, para la imagen X, cada *endmember* tenga asociado un mapa de abundancia.

Las etapas que componen la cadena de análisis son las siguientes: estimación del número de *endmembers*, reducción dimensional, extracción de *endmembers* y estimación de abundancias.

A modo de resumen, en la figura 2.9 se proporciona un diagrama con todas las etapas que componen la cadena [14]. Como se puede observar, en esta cadena aparecen tres etapas, que son las que se van a considerar obligatorias en este Proyecto Fin de Grado, si bien, para algunas aplicaciones, la etapa de reducción dimensional puede ser considerada como opcional. Sin embargo, este Proyecto la considera obligatoria porque, debido a los requisitos de tiempo real, esta etapa resulta indispensable. Por su parte, la etapa que se ha definido como estimación de *endmembers* no aparece porque su carácter es opcional, y así se ha considerado en este Proyecto.

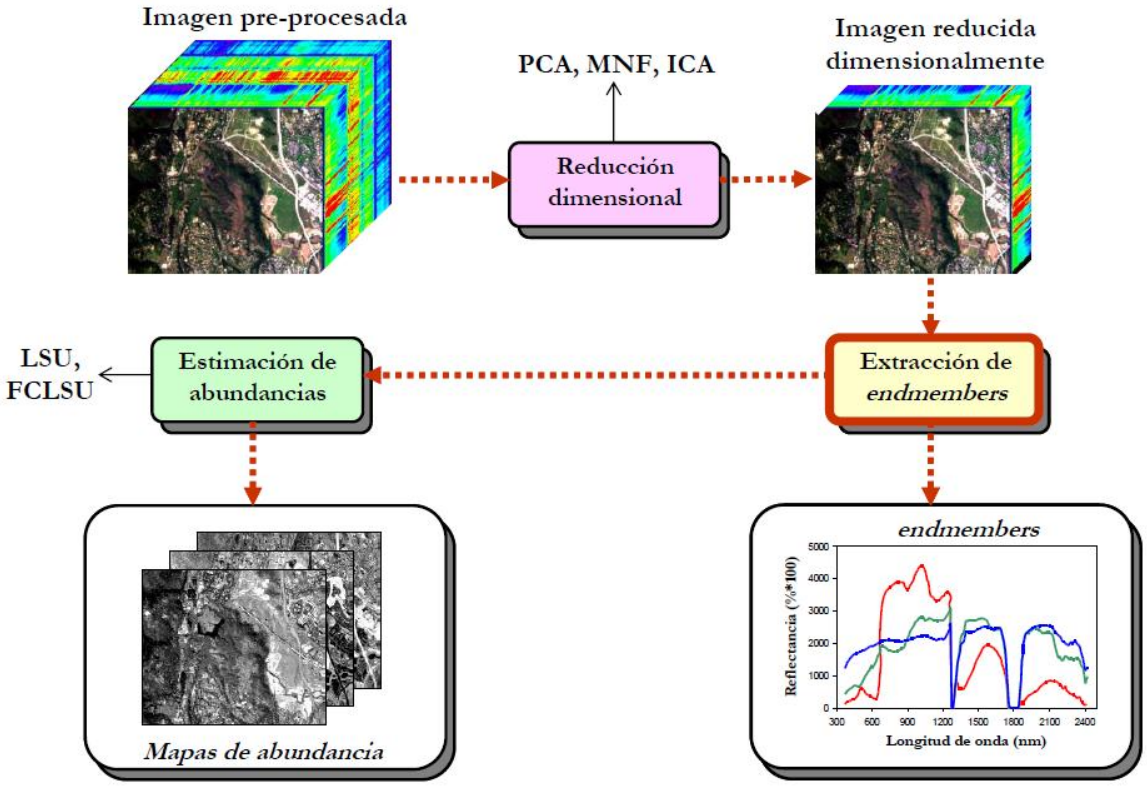


Fig. 2.9. Cadena de procesado de imágenes hiperespectrales (Fuente: http://www.umbc.edu/)

A continuación, se describirá cada una de las etapas mencionadas anteriormente, explicando la finalidad de cada una de ellas y mencionando algunos de los algoritmos que las implementan.

Estimación del número de *endmembers*

Esta fase es la primera en la cadena de desmezclado espectral. En esta etapa se realiza una primera estimación, que, partiendo de la imagen original *X*, proporciona el número de *endmembers* presentes en la imagen. Esta etapa parte de la premisa de que, en general, en las imágenes hiperespectrales el número de bandas -N- es mucho mayor que el número de materiales espectralmente distintos presentes en la imagen, lo que permite que dicha imagen pueda representarse en un subespacio de dimensionalidad menor que N [13]. La identificación de este subespacio permite llevar a cabo una reducción dimensional óptima - siguiente etapa de la cadena -, de tal forma que la cadena alcance su punto óptimo en lo referente a rendimiento, capacidad computacional y de almacenamiento.

Por tanto, esta etapa es opcional, puesto que su funcionalidad radica en complementar a la segunda fase y, consecuentemente, sólo se implementa en aquellas aplicaciones en las que estos parámetros constituyen puntos críticos.

A pesar de la importancia de esta etapa para aspectos como la velocidad de procesamiento o el tiempo de ejecución, su naturaleza opcional hace que la literatura actual no aporte un gran número de aproximaciones automáticas para el desarrollo de esta fase. En concreto, sólo existen dos aproximaciones ampliamente utilizadas: *VD* (de sus siglas en inglés, *Virtual Dimensionality*) y HYSIME (de sus siglas en inglés, *Hyperspectral Signal Identification with Minimum Error*) [15].

La primera de ellas se caracteriza por proporcionar cierta flexibilidad en la estimación, puesto que cuenta con un parámetro de entrada adicional que controla la sensibilidad; la segunda, por su parte, aporta funcionalidades avanzadas en lo referente al modelado del ruido existente en la imagen hiperespectral.

Reducción dimensional

Si bien ha quedado patente que una de las mayores ventajas de las imágenes hiperespectrales es que proporcionan la información suficiente y necesaria para discernir entre unos materiales y otros, lo cierto es que esto también puede derivar en varios problemas, como pueden ser el gran volumen de datos y la redundancia.

El primero de ellos - también conocido como *alta dimensionalidad* - es especialmente significativo cuando la capacidad computacional para procesar este volumen de información no es suficiente o cuando el tiempo supone una limitación crítica. Por su parte, el segundo problema deriva de que, en muchas ocasiones, el amplio número de bandas provoca una repetición de patrones y, en consecuencia, muchas de esas bandas aportan información redundante y, por tanto, irrelevante.

Teniendo todo esto en cuenta y, además, partiendo de la premisa expuesta en el apartado anterior, se puede afirmar que, al igual que en el caso anterior, en aquellas aplicaciones donde parámetros como la velocidad de procesamiento o la velocidad de ejecución sean críticos, la implementación de esta fase - en conjunto con la anterior - se hace indispensable.

En consecuencia, retomando el razonamiento anterior, el objetivo de esta fase es la identificación de aquél subespacio de dimensionalidad inferior a N tal que contenga la información relevante de la imagen hiperespectral, eliminando así ruido y datos redundantes de la misma.

Respecto a los algoritmos existentes en la literatura actual, cabe destacar que el abanico es amplio, si bien, de forma general, se puede afirmar que el algoritmo más ampliamente utilizado y, en consecuencia, más extendido, es el llamado *PCA* - de sus siglas en inglés, *Principal Component Analysis* -. De forma sencilla, se puede decir que este algoritmo constituye una técnica estadística que ordena los datos en función de su importancia, de tal forma que los primeros aportan la mayor parte de la información y, por el contrario, los últimos apenas contienen información relevante y, por tanto, pueden ser eliminados.

Además de este algoritmo, también se estudiarán otros, como el SPP - o *Spatial Pre-Processing* - o el MNF - *Minimum Noise Fraction*-. Cabe destacar que el último también ha sido desarrollado a partir del PCA.

Extracción de *endmembers*

El objetivo de esta fase es la identificación y extracción de las firmas espectrales características de cada uno de los *endmembers* presentes en la escena. De forma general, existen tres tipos de aproximaciones para resolver este problema: las aproximaciones estadísticas, las aproximaciones *sparse* y las aproximaciones geométricas [12].

La aproximación estadística se diferencia de las demás en que utiliza la información espacial en el proceso de extracción e identificación de *endmembers* - un buen ejemplo de ello podría ser la aproximación *Bayesiana*- [16]; por su parte, la aproximación *sparse* se diferencia de las demás en que, para la extracción de *endmembers*, utiliza librerías espectrales que contienen firmas espectrales reales y que no han de estar necesariamente presentes en la escena [17]. Esta decisión se fundamenta en la presunción de que dicha librería contiene las firmas espectrales de todos los *endmembers* que puedan aparecer en la escena, por lo que se puede eliminar la fase de identificación de *endmembers* y pasar directamente a la extracción de las firmas espectrales asociadas a esa imagen en particular. Sin embargo, no puede obviarse el hecho de que, en general, las firmas de la librería pueden no coincidir con las de la escena, puesto que las condiciones de adquisición son muy distintas: de forma general, las firmas de la librería se obtienen en un laboratorio en condiciones óptimas, mientras que la adquisición de datos está sujeta a una gran cantidad de condiciones variables, como las condiciones atmosféricas o de iluminación [11]. A este respecto, las otras aproximaciones presentan una ventaja, puesto que los *endmembers* se adquieren en las mismas condiciones que la propia imagen.

Respecto a las aproximaciones geométricas, éstas, a su vez, podrían dividirse en dos grupos: las que suponen que sí existen píxeles puros en la imagen y las que no. En ambos casos, los algoritmos asociados presentan una serie de ventajas y desventajas. Entre las desventajas más importantes, se encuentran las siguientes:

- Los algoritmos que no realizan esta suposición suelen llevar asociada la generación de firmas espectrales no necesariamente asociadas a materiales existentes en la naturaleza, pudiendo dar lugar a serios problemas a la hora de la interpretación física de estos *endmembers* [11].
- Por su parte, los que sí hacen esta suposición también presentan la desventaja de que, si no existen suficientes píxeles puros en la imagen, los algoritmos pueden fallar. No obstante, estas aproximaciones son las más estudiadas y documentadas en la literatura de desmezclado espectral, por lo que, en general, son las que más se han utilizado hasta el momento [12]. Entre ellos, se pueden encontrar los siguientes: N-FINDR (*N-finder algorithm*), VCA (*Vertex Component Analysis*) e IEA (*Iterative Error Analysis*).

Por último, cabe destacar que, a lo largo de los últimos años, se han desarrollado una serie de algoritmos que aúnan las características de las aproximaciones geométricas y espaciales, puesto que utilizan, de forma conjunta, la información espacial y espectral. Los algoritmos más relevantes de esta categoría son SSEE (*Spatial-Spectral Endmember Extraction*) y AMEE (*Authomatic Morphological Endmember Extraction*) [12, 16].

En este Proyecto Fin de Grado se va a realizar un estudio de los algoritmos más utilizados hasta el momento, comparando ventajas y desventajas para, finalmente, implementar el que parezca más completo. En concreto, los algoritmos que se van a analizar son los siguientes: OSP (*Orthogonal Subspace Projection*), N-FINDR, SSEE, AMEE, VCA e IEA.

Estimación de abundancias

Esta es la última fase de la cadena de procesado de una imagen hiperespectral, y consiste en la obtención y representación, para cada píxel, de las fracciones de abundancia asociadas a cada *endmember* presente en el mismo. Como ya mencionamos a lo largo del presente capítulo, la forma más habitual de representar esta información es mediante los llamados *mapas de abundancias*, de tal forma que se ha de generar uno por cada *endmember* existente en la escena.

Así, el mapa de abundancia del *endmember* β será un mapa de bits en el que el valor de cada píxel representa la proporción en la que dicho *endmember* β está presente en el mismo, de tal forma que se asocie un valor extremo - por ejemplo, 255 o blanco absoluto - al mayor grado de coincidencia, y el otro valor extremo - en este caso, 0 o negro absoluto - al caso contrario.

Enlazando con las ideas expuestas en apartados anteriores, en consecuencia, en el caso de que un píxel sea considerado puro - esto es, que sólo aparezca un *endmember* en dicho píxel -, en él se debe mostrar un predominio absoluto con respecto a los otros *endmembers* en el mapa de abundancia asociado al mismo. En otras palabras, si, por ejemplo, existiese un píxel puro, δ, del *endmember* β, en el mapa de abundancia correspondiente a dicho endmember, ese píxel debería aparecer completamente blanco. Si, por el contrario, existieran píxeles puros de otros *endmembers* distintos del *endmember* β, éstos deberían aparecer completamente negros, como le ocurriría al píxel δ si se cambiase al mapa de bits de otro *endmember*.

De forma genérica, esta representación supone realizar el paso inverso a la descomposición del modelo lineal de mezcla visto anteriormente. En la figura 2.7 veíamos que el modelo lineal de mezcla partía de la presunción de que cada píxel estaba formado por una combinación lineal de las firmas espectrales de los distintos *endmembers* presentes en la escena, ponderadas por su abundancia asociada. En consecuencia, la cadena de desmezclado tenía como objetivo obtener, por un lado, esas firmas espectrales y, por otro, esas abundancias asociadas. En este caso, la creación de los mapas de abundancia supone volver a generar esa combinación lineal pero, en lugar de sumar todas las contribuciones de cada *endmember* en un único píxel, representar cada una por separado.

Continuando con la nomenclatura mostrada en la figura 2.7, si la composición del píxel inicial, *x*, podía expresarse como aparece en la ecuación 2.2:

$$x = a_1 \cdot m_1 + a_2 \cdot m_2 + a_3 \cdot m_3 \tag{2.2}$$

Entonces, la representación de las abundancias supondrá lo siguiente:

$$x_1 = a_1 \cdot m_1 \qquad x_2 = a_2 \cdot m_2 \qquad x_3 = a_3 \cdot m_3 \tag{2.3}$$

Donde x_1, x_2 y x_3 representan el mismo píxel, pero para distinto mapa de abundancia, de tal forma que el píxel x_n forma parte del mapa de abundancia del *endmember* m_n.

Respecto a los distintos algoritmos existentes para implementar esta fase, en este Proyecto Fin de Grado se estudiarán los más habituales en la literatura de imágenes hiperespectrales, que son: UCLS (*UnConstrained Least Squares*, también denominado LSU), SCLS (*Sum-to-one Constrained Least Squares*), NCLS (*Non - negativity Constrained Least Squares*), FCLS (*Fully Constrained Least Squares*) e ISRA (*Image Space Reconstruction Algorithm*) [11].

A modo de resumen, en la figura 2.10 se proporciona un diagrama con los distintos algoritmos para cada fase, mostrando las posibles combinaciones que dan lugar a distintas cadenas de análisis de imágenes hiperespectrales. El denominador común de todas las posibles cadenas es la interfaz de entrada y salida, puesto que todas parten de la imagen hiperespectral original y generan, a su salida, las firmas espectrales de los *endmembers* presentes en la escena y sus mapas de abundancia asociados.

Como se puede observar, las posibilidades de combinación son muy elevadas, por lo que, en capítulos posteriores, se analizará la bondad de cada uno de los algoritmos y, para cada una de las etapas descritas, se elegirá aquél que mejores prestaciones proporcione respecto a los criterios que se especifiquen. Posteriormente, los algoritmos elegidos serán implementados en el lenguaje establecido en el capítulo 1 y, en consecuencia, se dispondrá de una cadena de procesamiento completa.

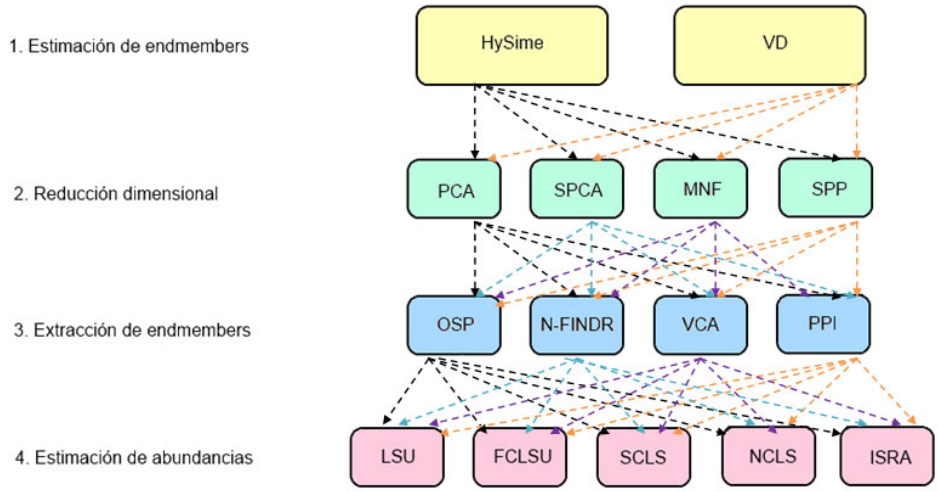


Fig. 2.10. Esquema resumen de las distintas cadenas de procesado de una imagen hiperespectral

2.2. Imágenes hiperespectrales: aplicaciones

Ahora que han sido explicados los conceptos básicos relacionados con las imágenes hiperespectrales, en este apartado se va a proporcionar un resumen de los distintos ámbitos de aplicación de esta tecnología. Conviene destacar que se hará especial hincapié en las aplicaciones médicas, puesto que, como se mencionó en el capítulo 1, este documento se inscribe dentro de un proyecto europeo orientado a la aplicación de esta tecnología a la detección del cáncer.

Como se mencionó en el apartado anterior, el desarrollo de esta tecnología surgió como consecuencia de la evolución de la teledetección o *remote sensing*, y por tanto, sus aplicaciones iniciales estaban enfocadas a la observación terrestre y la geología. Sin embargo, con el paso de los años este horizonte se ha ido ampliando y, en la actualidad, el rango de aplicaciones de la espectroscopia de imagen (o *hyperspectral imaging*) se ha ampliado considerablemente. Algunos de los campos de aplicación más relevantes son: observación terrestre, seguridad y Defensa, agricultura e ingeniería de alimentos, gestión de desastres, monitorización del medio ambiente, ciencia forense y verificación de documentos, análisis microscópico, farmacología y medicina [18, 21, 25, 28, 33, 37, 40, 47][3].

[3]Estas y otras categorías pueden consultarse en:
Headwall Photonics: http://www.headwallphotonics.com/applications/

A continuación, se hará una breve descripción de cada área de aplicación, explicando la forma en la que las imágenes hiperespectrales han servido de ayuda en esos campos y aportando, en la medida de lo posible, ejemplos significativos de ello.

Observación terrestre

Como se ha mencionado anteriormente, las aplicaciones iniciales de la tecnología de imagen hiperespectral estaban directamente enfocadas a la observación terrestre. Uno de los primeros usos de esta tecnología fue el de la identificación y localización eficiente de los materiales en la escena, gracias a la obtención de imágenes hiperespectrales a través de sensores aerotransportados. Esto hizo posible que, por ejemplo, sectores como el de la mineralogía o industrias como la minera y la petrolífera se vieran directamente beneficiadas, puesto que pudieron servirse de ello para la búsqueda y localización de nuevos yacimientos y minas.

Al ser una de las primeras aplicaciones en desarrollarse, la identificación de minerales a través de imágenes hiperespectrales es una de las aplicaciones más documentadas [2, 6, 19], por lo que no se ahondará en ello. Sin embargo, conviene destacar que, en los últimos años, la teledetección no sólo se ha dedicado a la observación de la superficie terrestre, sino que se ha empezado a emplear en otros planetas, como Marte [20]. En [20], [21] y [22] se aportan sendos estudios sobre los minerales existentes en la superficie de Marte. Su objetivo es el análisis de estos materiales para tratar de fundamentar la posibilidad de existencia de agua en dicho planeta en algún momento de su historia. En concreto, en [21] se realiza un estudio de algunos de los datos obtenidos por el CRISM[4] (de sus siglas en inglés, *Compact Reconnaissance Imaging Spectrometer for Mars*), que es un sensor que obtiene imágenes cuyo rango espectral varía entre 0.37–3.92 μm, captando, en consecuencia, información del espectro visible y del infrarrojo cercano. Este sensor fue lanzado en 2005 con el objetivo de documentar la existencia de agua en Marte en el pasado, basándose en las propiedades de los minerales observados, así como cartografiar y caracterizar la composición y geología de la superficie del planeta [22].

La hipótesis de la que se partía se basaba en la demostración de la existencia de *filosilicatos* en la superficie de Marte, al ser éstos un tipo de mineral que necesita de la existencia de agua durante un período prolongado de tiempo para su formación [23]. Por tanto, si las imágenes conseguidas por el sensor consiguiesen extraer la firma asociada a este material, podría probarse que, como mínimo, existió agua en Marte durante el período de tiempo en que estos minerales se formaron.

En la figura 2.11 se puede observar el mapa de abundancias elaborado en base a las imágenes obtenidas por el sensor CRISM en enero de 2007[5]. A la izquierda, se puede observar la situación geográfica de la escena captada (al oeste de la zona conocida como *Mawrth Vallis*). A la derecha, se muestran los distintos grados de coincidencia de ciertos filosilicatos en la superficie analizada: en color azul se muestran aquellos que contienen aluminio y en distintas tonalidades de magenta aparecen aquellos que contienen hierro y magnesio.

En consecuencia, los datos obtenidos gracias a este sensor parecen demostrar la existencia de grandes masas de agua en la superficie de dicho planeta, además de revelar una complejidad sorprendente en la geología del mismo.

Resonon: http://www.resonon.com/applications_main.html
[4] http://crism.jhuapl.edu/
[5] http://crism.jhuapl.edu/gallery/featuredImage/image.php?page=1&gallery_id=2&image_id=218

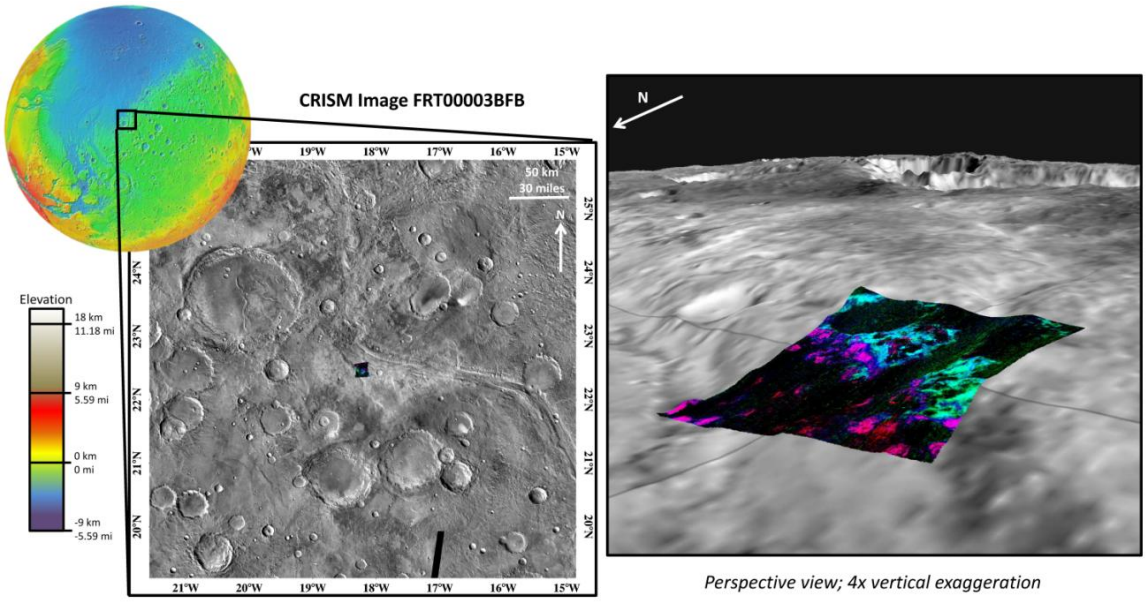


Fig. 2.11. Evidencia de la existencia de filosilicatos en Marte (Mawrth Vallis)
(Fuente: http://crism.jhuapl.edu/gallery/featuredImage/index.php)

Seguridad y Defensa

Además de la teledetección y la observación terrestre, otro campo que impulsó el desarrollo y evolución de esta tecnología fue el de la seguridad y la defensa, puesto que, antes de comercializarse, esta tecnología se desarrolló para aplicaciones militares. Algunas de las aplicaciones más conocidas son la localización y detección de blancos y la prevención de ataques químicos.

La localización y detección de blancos se centra, sobre todo, en proporcionar una herramienta sustitutiva de la detección por imágenes térmicas en aquellas situaciones en las que su aplicación no arroja resultados satisfactorios como, por ejemplo, cuando el contraste térmico es bajo. En dichas situaciones, varios estudios han probado que, efectivamente, la aplicación de esta tecnología a la detección de objetivos proporciona resultados satisfactorios [24, 25].

Por su parte, la prevención de ataques químicos se centra, en gran medida, en la detección remota de los llamados *Chemicals Warfare Agents* (CWA o Agentes de Guerra Química) y los *Toxic Industrial Compounds* (TIC o Compuestos Industriales Tóxicos). Estas sustancias son prácticamente invisibles y, en consecuencia, muy difíciles de detectar; además, su toxicidad es tal que suponen uno de los ataques más graves que pueden desencadenarse en un contexto militar – por ejemplo, sobre tropas de tierra –. Sin embargo, estos ataques no suponen sólo una amenaza para el colectivo militar, sino también para el civil, por lo que durante los últimos 30 años se han estudiado posibles soluciones para la detección de este tipo de sustancias [26]. Según Manolakis [26], la mayoría de ellas se basan en el desarrollo de sensores aerotransportados que detecten posibles amenazas, identifiquen estos agentes químicos, cuantifiquen su presencia, localicen el foco de fuga o liberación y que, además, realicen todas estas operaciones con un tiempo de respuesta tal que permita una rápida acción sobre el problema. Por ejemplo, algunos de los sensores desarrollados para este tipo de misiones fueron el *FIRST* y el *Hyper-Cam*, ambos desarrollados por la compañía canadiense *Telops Inc*. Según

los datos proporcionados por dicho fabricante[6], este último sensor permite la identificación y localización de gases desde distancias superiores a 5 kilómetros y, en total, sus distintos modelos proporcionan imágenes con un rango espectral entre 1.5 – 11.8 μm, con una resolución espectral superior a 0.25 cm^{-1}.

Agricultura e ingeniería de alimentos (*food engineering*)

Otros sectores que se han visto beneficiados con el desarrollo de esta nueva tecnología son la ingeniería de alimentos y la agricultura. A continuación, se describe brevemente el impacto de las imágenes hiperespectrales en ambos campos, proporcionando ejemplos ilustrativos.

En la industria alimentaria, la aplicación de las mismas ha mejorado considerablemente los controles de calidad y seguridad. Las cadenas tradicionales de análisis de calidad aplicaban técnicas que conllevaban un gran tiempo de procesado, eran muy costosas y, lo más importante, implicaban la destrucción de las muestras analizadas. Esto ha abierto un campo de investigación que busca la optimización de estas cadenas, siendo especificaciones indispensables la rapidez, la precisión y la conservación de las muestras utilizadas. En este sentido, la espectroscopia de imagen ofrece una gran cantidad de posibilidades, puesto que es una técnica no invasiva – y, en consecuencia, no destructiva – que, al proporcionar información espectral y no únicamente espacial, puede extraer información no registrada en el espectro visible. A este respecto, en los últimos años se han realizado diversos estudios sobre la aplicación de esta metodología a la detección de defectos – como golpes o contaminantes - en la superficie de alimentos. Por ejemplo, en 2005 se desarrolló un sistema para la detección de golpes en manzanas, llegando a obtener un porcentaje de eficiencia del 87% [27]. Asimismo, según [28], Gómez-Sanchis et al. [29] demostraron un año antes "*el potencial de esta tecnología para la detección de las infecciones causadas por la bacteria* Penicillium digitatum *en frutas cítricas antes de que fuera detectable en inspecciones humanas*", consiguiendo un 80% de efectividad. Además, este grupo ha continuado su investigación y, en [30] aportan un estudio sobre la aplicación de un sistema de imágenes hiperespectrales para la detección de mandarinas infectadas por esta misma bacteria.

Por su parte, la agricultura también ha abierto nuevos campos de investigación gracias a la aplicación de imágenes hiperespectrales. La mayoría de estas aplicaciones tienen la misma finalidad que las de la industria alimentaria – control de calidad y detección de desperfectos o, en este caso, plagas en cultivos), por lo que no se va a entrar en detalle en ellas. No obstante, algunos de estos campos de investigación sí resultan novedosos; es el caso de la detección de proteínas animales en piensos compuestos para evitar el desarrollo de enfermedades derivadas de ellas, como la *encefalopatía espongiforme bovina*, popularmente conocida como la *enfermedad de las vacas locas*. Parece ser que el agente transmisor de esta enfermedad se encuentra en unas sustancias utilizadas como complemento alimenticio en los animales de granja: las llamadas proteínas animales procesadas (PAP, del inglés *Processed Animal Proteins*) y, en concreto, en las MBM (*Meat and Bone Meals*), puesto que, tras ser prohibidas para uso alimenticio por la Unión Europea, los casos de esta enfermedad se redujeron notablemente [31]. En [32] se proporciona un estudio sobre la utilización de imágenes hiperespectrales para la detección de estas sustancias en piensos compuestos. En concreto, el estudio presenta un sistema automático para la detección de la presencia de este contaminante en cantidades cercanas al 1%.

[6] http://www.telops.com/en/products/hyperspectral-imaging/607-hyper-cam55

Gestión de desastres (*disaster management*)

En los últimos años, las nuevas posibilidades proporcionadas por la observación terrestre a través de imágenes hiperespectrales han dado lugar a nuevos campos de aplicación, como el de la gestión de desastres. Este nuevo campo se basa en la utilización de dicha tecnología para la extracción de una serie de parámetros y características que permitan, entre otras posibilidades, una rápida actuación frente a la ocurrencia de desastres naturales, con el objetivo de paliar los efectos de los mismos de forma óptima. La observación terrestre y, en concreto, las imágenes hiperespectrales se han aplicado a desastres naturales de distinta índole; por ejemplo, en [33] se realiza un análisis sobre la aplicación de la teledetección por satélite para el estudio y valoración de los riesgos asociados a desastres naturales como terremotos, erupciones volcánicas, desbordamientos, corrimientos de tierras e inundaciones en zonas costeras.

En concreto, existen numerosos artículos en la literatura de las últimas décadas sobre la búsqueda y localización tanto de filtraciones como de vertidos petrolíferos [34, 35, 36, 37]. Como ejemplo, en este apartado se pueden citar algunas de las investigaciones desarrolladas tras uno de los mayores desastres ecológicos de la historia de España: el hundimiento del buque petrolero *Prestige* frente a las costas de Galicia (España) en noviembre de 2002. Gracias a la rápida respuesta de dos compañías de investigación especializadas en oceanografía y teledetección de gran resolución – *AvelMor* y *Borstad Associates Ltd.* -, dos semanas después del hundimiento del petrolero se adquirió un conjunto de imágenes hiperespectrales para determinar el mapa de alcance del vertido petrolífero. Los mapas extraídos incluyeron información precisa de las superficies contaminadas, llegando a identificar superficies menores de 5 metros cuadrados e, incluso, zonas afectadas a nivel submarino, a pocos metros de la superficie del mar [38, 39].

En la figura 2.12 se aporta, a modo explicativo, los resultados obtenidos durante estas investigaciones para dos imágenes hiperespectrales. En la figura de la izquierda se observa un vertido que se extiende por una superficie de gran extensión, mientras que en la figura de la derecha se observan varios vertidos, pero de extensión muy reducida. Además, junto a cada imagen se puede observar el mapa de abundancia generado, en el que el color rojo se corresponde con un grado de coincidencia máximo y el color negro, con un grado de coincidencia mínimo. En ella se puede observar que el vertido de más extensión se detecta a la perfección; por su parte, los vertidos más pequeños no son detectados al 100%, puesto que, como puede apreciarse, sólo se detectan tres de los cuatro vertidos. No obstante, esto puede catalogarse como buenos resultados, ya que se obtiene un 75% de efectividad en superficies cuya extensión es inferior a los 5 metros cuadrados.

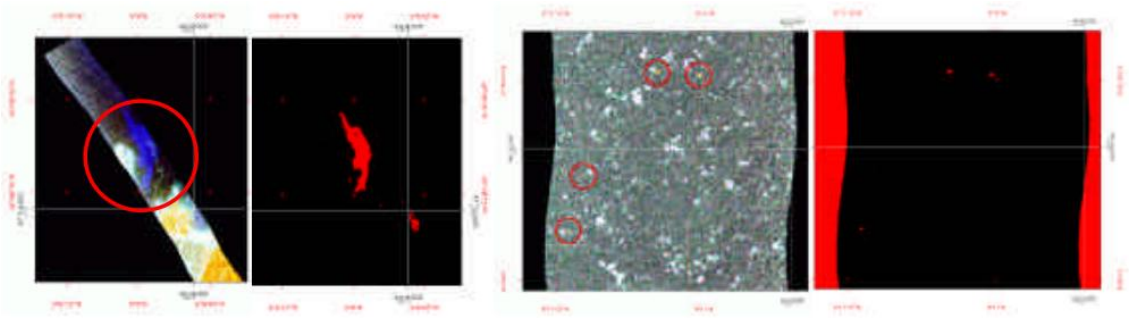


Fig. 2.12. Imágenes originales y mapas obtenidos tras el hundimiento del Prestige. Vertido de gran tamaño (izquierda) y vertidos de pequeña o muy pequeña superficie (derecha)
(Fuente: http://www.earsel.org/)

Monitorización del medio ambiente (*environmental monitoring*)

La monitorización del medio ambiente engloba todos los procesos y actividades encaminados a monitorizar la calidad del medio ambiente. Los ámbitos de monitorización, en general, son: calidad del aire, calidad del suelo y calidad del agua.

La monitorización de la calidad del aire se centra, principalmente, en la vigilancia de la fluctuación de los niveles de polución y, en concreto, en la medición de las emisiones de sustancias contaminantes, tales como monóxido (CO) y dióxido de carbono (CO_2), dióxido de azufre (SO_2), dióxido de nitrógeno (NO_2) y otros gases procedentes de las emisiones tanto de vehículos como de zonas industriales [40]. A este respecto, en [41] se proporciona un ejemplo de aplicación de esta tecnología a la detección de sustancias contaminantes en una planta petroquímica en el sur de Texas. Los primeros resultados localizaban correctamente la existencia de varias fugas, e identificaban sustancias como benceno, amoniaco y otros compuestos orgánicos. Por su parte, en [42] se proporciona un estudio piloto sobre la detección de CO_2 en zonas subterráneas de poca profundidad; en él, también se analizan los distintos patrones que sigue este compuesto al filtrarse por suelo, agua y aire, así como el efecto que tiene en la salud de la vegetación de los alrededores. Para llevar a cabo este estudio, construyeron un dispositivo que liberaba CO_2 de forma controlada y lo enterraron en un pozo a poca profundidad en Montana (EE.UU.), de tal forma que se simulaba la liberación natural de CO_2 a través del suelo. Sobrevolaron la zona con el *Resonon airborne hyperspectral imaging system*[7], que midió el espectro de reflectancia de la misma zona de vegetación casi diariamente, antes, durante y después de una de las fugas controladas. Las firmas espectrales de dicha vegetación mostraron claramente un efecto negativo en las plantas, especialmente cuando la concentración de CO_2 del suelo alcanzó cotas situadas entre el 4% y el 8%, lo que suponía entre el doble y el triple del nivel existente antes de la liberación de dicho compuesto.

Por su parte, la monitorización de la calidad del suelo se centra en el análisis del estado del mismo en una determinada zona, de tal forma que se pueda comprobar si ese suelo está en las óptimas condiciones para proporcionar sustento a su ecosistema natural. En otras palabras, la monitorización del suelo proporciona información acerca del grado de bondad del mismo para mantener la biodiversidad y productividad. Las aplicaciones de monitorización del suelo para la medición de su calidad están profundamente relacionadas con la agricultura; como esta sección ya ha sido explicada anteriormente, no se profundizará en ella. Estas aplicaciones se basan en la utilización de sensores hiperespectrales para la monitorización de cultivos, de tal forma que estudiando parámetros como la cantidad de agua, el grado de riqueza del suelo en nutrientes o la existencia de plagas o enfermedades se pueda elaborar un plan de acción y mejorar la calidad y la productividad de los cultivos. Algunos ejemplos de ello pueden encontrarse en [43, 44].

Por último, la monitorización de la calidad del agua consiste, al igual que las anteriores, en la medición de una serie de parámetros que permitan dilucidar si la calidad del agua es lo suficientemente buena para los usos que se le quiera dar - consumo humano, riego, hábitat de ciertas especies, etc -. Sin embargo, muchas de las técnicas existentes hasta el momento son invasivas. A este respecto, la tecnología de imágenes hiperespectrales es la más destacada, puesto que, como ya se ha mencionado anteriormente, supone un método no invasivo de análisis y control. Por ejemplo, en [45] se proporciona un estudio sobre el análisis de la calidad del agua en una bahía cerca de Brisbane, Australia. Para ello, se tomaron medidas con el sensor

[7] http://www.resonon.com/application_ecology.html

Hyperion[8], que fue el primer sensor hiperespectral - de alta resolución - puesto en órbita. Anteriormente, todos los sensores hiperespectrales estaban diseñados para ser aerotransportados, es decir, a medir desde una distancia equiparable a la del vuelo de un avión. Sin embargo, en 2000, cuando se lanzó *Hyperion*, se empezaron a tomar medidas desde la órbita terrestre. Este hecho mejoró enormemente la calidad de las imágenes tomadas por los satélites, puesto que, hasta el momento, éstas se generaban con una resolución muy baja -en comparación con las imágenes hiperespectrales-. En dicho estudio, estas imágenes fueron utilizadas para comprobar el grado de eficacia del sensor para localizar ciertas sustancias en el agua australiana, comparándolas con un análisis realizado in situ - aplicando la metodología clásica-. Los resultados fueron muy satisfactorios: las concentraciones de sustancias como materia orgánica o clorofila medidas con el sensor *Hyperion* se correspondían con las estimadas en los estudios clásicos que se realizaron durante los mismos días en que las imágenes fueron capturadas.

Ciencia forense y verificación de documentos (*forensics and document verification*)

Además de las aplicaciones ya mencionadas, durante los últimos años la utilización de las imágenes hiperespectrales ha tomado fuerza en otros campos radicalmente distintos a los ya descritos. Este es el caso, por ejemplo, de campos como la ciencia forense o la conservación y verificación de documentos históricos y obras de arte.

Respecto al campo de la ciencia forense, en los últimos años ha cobrado fuerza la aplicación de las imágenes hiperespectrales a la obtención y análisis de pruebas. Esto se debe a la naturaleza no invasiva de las mismas, puesto que no dañan la prueba original y, además, proporcionan información objetiva [46]. Algunas de las áreas de investigación forense donde la tecnología de imágenes hiperespectrales puede ser aplicada son: visualización de restos de sangre, comparación de fibras, visualización de restos de pólvora, intensificación y extracción de huellas dactilares latentes, análisis de documentos y análisis de residuos inflamables[9].

Para aportar un ejemplo ilustrativo, en la figura 2.13 se observa el proceso de obtención de una huella dactilar latente. A la izquierda, se puede apreciar el resultado obtenido tras aplicar técnicas convencionales de extracción de huellas; por su parte, a la derecha aparece el resultado tras aplicar técnicas de imágenes hiperespectrales. Como puede comprobarse, el resultado es sustancialmente mejor en esta última, puesto que la huella puede apreciarse con total claridad.

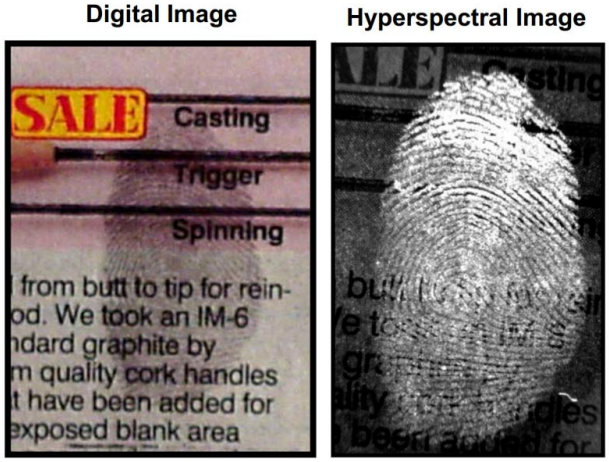


Fig. 2.13. Obtención de una huella dactilar aplicando técnicas de imágenes hiperespectrales

[8] http://eo1.usgs.gov/sensors/hyperion
[9] http://www.chemimage.com/webinars/HSI-Basics-6-14-11/CI_Intro_HSI_Webinar0614.pdf

Por su parte, el campo de la verificación de documentos históricos - o conservación - y obras de arte se ha visto beneficiado con esta nueva tecnología, ya que extiende de forma considerable las posibilidades de la *reflectografía infrarroja*[10], que es una técnica que, mediante luz infrarroja, atraviesa las capas visibles y permite visualizar las subyacentes sin dañar la superficie. La combinación de esta tecnología con la espectroscopia de imagen ha sido aplicada de forma satisfactoria tanto a pinturas como al estudio de documentos históricos. A pesar de que el desarrollo de este nuevo ámbito de aplicación es reciente, se pueden clasificar las distintas líneas de actuación existentes en las siguientes categorías: estudio de materiales de valor arqueológico, cultural o histórico, evaluación y monitorización de métodos de conservación de obras de arte y digitalización de imágenes para documentación y archivo [47].

A este respecto, quizá uno de los ejemplos más significativos y reveladores es el del *Palimpsesto de Arquímedes*. Este famoso documento, que originalmente contenía una copia de diversas obras de Arquímedes, fue escrito durante el siglo X y, dos siglos después, fue borrado y reescrito con oraciones y salmos. Para ello, cada página fue cortada a la mitad, girada 90º y reutilizada [48]. Gracias a la aplicación de diversas técnicas de espectroscopia de imagen, durante los últimos años se ha podido extraer el texto original de gran parte del manuscrito. En concreto, uno de los logros de la tecnología de imágenes hiperespectrales tuvo lugar en 2009 cuando, dos años después de ser anunciado que se había encontrado un nuevo texto en el palimpsesto, éste fue recuperado en su totalidad gracias a la aplicación de algoritmos como el PCA. Este nuevo texto era un comentario sobre una obra de Aristóteles presuntamente atribuido a Alejandro de Afrodisias [49, 50].

En la figura 2.14 se puede observar un fragmento del palimpsesto en el que se puede apreciar los resultados que proporciona el PCA. En la figura 2.14 (a) aparece el fragmento bajo luz visible; aparentemente, no se observan otros trazos que no sean los verticales. En la figura 2.14 (b) se muestra el mismo fragmento, pero bajo iluminación infrarroja y, en esta ocasión, ya se pueden discernir unos trazos horizontales, que aparecen en color blanco. Por último, en la figura 2.14 (c) aparece, de nuevo, el mismo fragmento, pero tras ser procesado por el algoritmo anteriormente citado. En esta ocasión, el texto original queda de manifiesto, llegando incluso a ser comprensible a simple vista.

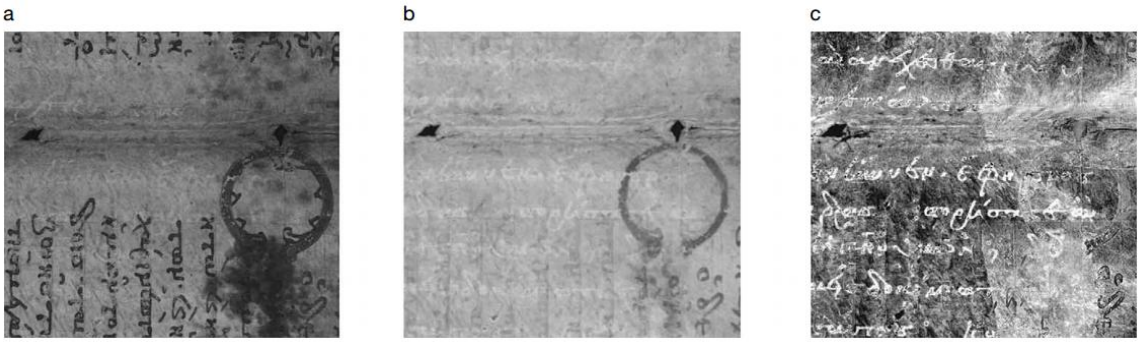


Fig. 2.14. Comparativa de un fragmento del palimpsesto. (a) Apariencia del fragmento con luz visible (b) Apariencia con luz infrarroja (c) Resultado tras aplicar PCA (Copyright perteneciente al dueño del Palimpsesto de Arquímedes) [35]

[10] http://www.museothyssen.org/microsites/tintoretto/estudio_mediante_reflectografia_infrarroja.html

Análisis microscópico y farmacología

Uno de los últimos sectores en beneficiarse de la tecnología de imágenes hiperespectrales ha sido el del análisis microscópico; desde principios de este siglo, muchas líneas de investigación se han centrado en el desarrollo de sensores hiperespectrales capaces de tomar imágenes a nivel microscópico, que han recibido el nombre de *hyperspectral imaging microscopes* o, en español, microscopios de imágenes hiperespectrales. Por ejemplo, en [51] se explica el desarrollo y comprobación del funcionamiento de uno de estos nuevos *microscopios*: en concreto, el dispositivo descrito consta, básicamente, de un microscopio estándar al que se le ha acoplado un sensor hiperespectral, y del que se graba la salida con una cámara CCD (*charge-coupled device*)[11].El desarrollo de este nuevo tipo de sensores ha provocado el nacimiento de una gran cantidad de aplicaciones en varios sectores, como el farmacéutico. Este sector ha aplicado, satisfactoriamente, estos microscopios al análisis de calidad de sus productos. Por ejemplo, en [52] son capaces de solucionar dos problemas: en primer lugar, pueden detectar contaminantes en la superficie de ciertas píldoras - en este caso, la presencia del colorante *carmín de índigo*-; en segundo lugar, tras analizar seis tipos de pastillas, de las cuales tres presentaban buenas propiedades de disolución y las otras tres no, son capaces de identificar la fuente del problema, que radica en la distribución superficial de una sustancia denominada *estearato de magnesio*. Para llegar a esta conclusión, han observado que la proporción de esta sustancia es mayor en las píldoras con peores propiedades disolutivas. Además, han tenido en cuenta que esta sustancia es hidrófoba; eso implica que, ante una mayor concentración en la superficie, peor serán las propiedades disolutivas.

La figura 2.15 muestra las imágenes obtenidas de las seis pastillas para cuatro tipos distintos de sustancias analizadas, entre las que se encuentra el citado estearato de magnesio - figura 2.15 (d)-. En ella se puede observar que, efectivamente, ésa es la sustancia que mayor disparidad presenta en cuanto a la proporción en cada tipo de pastilla, puesto que los espectros de las otras tres figuras presentan más similitudes. Además, se puede apreciar que, en el caso de las pastillas con malas propiedades disolutivas, la concentración del estearato es mayor.

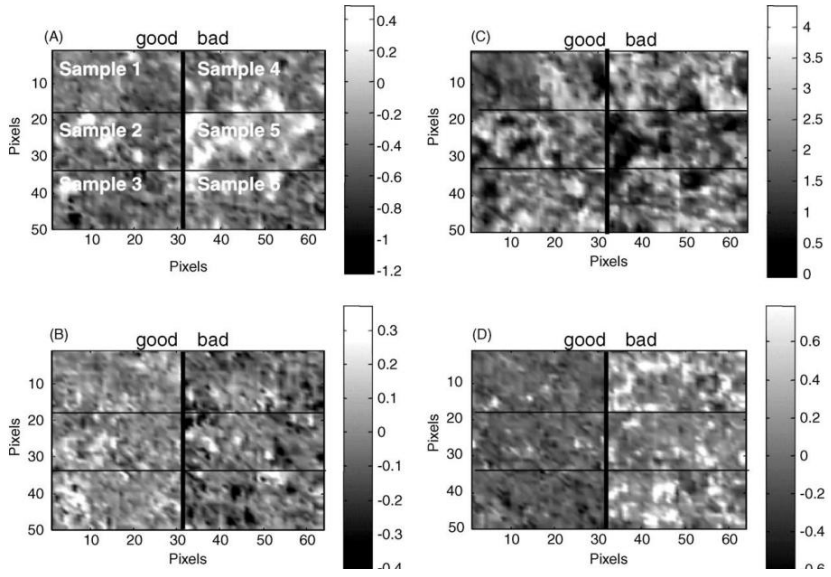


Fig. 2.15. Imagen hiperespectral para una banda específica. (A) Celulosa microcristalina (B) Ingrediente activo (C) Poloxámero (D) Estearato de magnesio [52]

[11] http://www.microscopyu.com/articles/digitalimaging/ccdintro.html

Medicina

Si se tuviese que citar uno de los campos en los que la tecnología de imágenes hiperespectrales ha tenido un mayor impacto, sin duda ése sería el de la medicina. La aplicación de esta tecnología a nivel microscópico ha supuesto la consecución del análisis de imágenes captadas a nivel molecular. Esto, unido a la presunción generalizada de que las características de fluorescencia, absorción y dispersión de la luz de los tejidos cambian según avanzan las enfermedades [53], ha supuesto una auténtica revolución en el mundo de la medicina, abriendo un nuevo mundo de posibilidades. En [53] se proporciona un amplio análisis de la literatura relacionada con las aplicaciones médicas de las imágenes hiperespectrales - *medical hyperspectral imaging* o MHSI - desde 1988 hasta 2013, repasando los logros conseguidos y señalando los posibles trabajos futuros en este campo. Este artículo divide los campos de aplicación médicos en dos grandes ramas: la ayuda a la detección y diagnóstico de enfermedades y la asistencia a los cirujanos durante las intervenciones quirúrgicas. A su vez, en cada una de estas ramas se destacan las líneas de actuación más importantes, que son:

- Detección y diagnóstico de enfermedades: En esta rama, se estudia la detección y diagnóstico de enfermedades cardíacas y circulatorias, enfermedades retinales, pie diabético[12], shocks, isquemias y, sobre todo, distintos tipos de cáncer. Entre las distintas tipologías de cáncer existentes, las imágenes hiperespectrales han demostrado un gran potencial para la diagnosis de cáncer de cérvix, mama, colon, piel, tráquea, ovario, próstata, esófago, lengua y cerebro.
- Asistencia a la cirugía: La tecnología de imágenes hiperespectrales ha sido aplicada para la asistencia de intervenciones como mastectomías, cirugía vesicular, cirugía renal y cirugía abdominal.

Sin duda alguna, el reto más revolucionario y ambicioso es el de utilizar las imágenes hiperespectrales para la detección de tumores cancerígenos, así como para guiar a los cirujanos durante la extirpación de los mismos. Como se ha mencionado anteriormente, el fundamento de esta nueva rama de estudio es el hecho de que los cambios morfológicos y bioquímicos que se producen como consecuencia del desarrollo de distintos tipos de tumores a nivel celular alteran las características ópticas de los tejidos y, en consecuencia, pueden proporcionar información útil para la localización y diagnóstico de tumores, midiendo los niveles de ciertos *biomarcadores*[13], tales como la concentración de hemoglobina o la saturación de oxígeno en sangre [53, 54, 55].
A modo de ejemplo, en la figura 2.16 se muestran varias imágenes en las que se puede apreciar la saturación de oxígeno de, en este caso, la retina. En (a) se muestra el mapa de saturación de un varón de 28 años, y en (b) se muestra la imagen normal; lo mismo ocurre en (c) y (d), pero para una muestra tomada de un varón de 58 años. Como puede observarse en ellas, las zonas con mayor presencia de oxígeno aparecen marcadas con tonalidades azules. Esto permite, a simple vista, una diferenciación de, por ejemplo, el tipo de vaso conductor: se sabe que las arterias son las que llevan la sangre oxigenada al resto de las células del cuerpo, mientras que las venas devuelven la sangre, que ya no está oxigenada, al corazón; en consecuencia, los vasos que aparecen marcados en azul se corresponden con las arterias, puesto que su concentración de oxígeno es mayor.

[12] Esta enfermedad es una patología derivada de un empeoramiento severo en pacientes con diabetes: http://es.wikipedia.org/wiki/Pie_diab%C3%A9tico
[13] http://es.wikipedia.org/wiki/Biomarcador

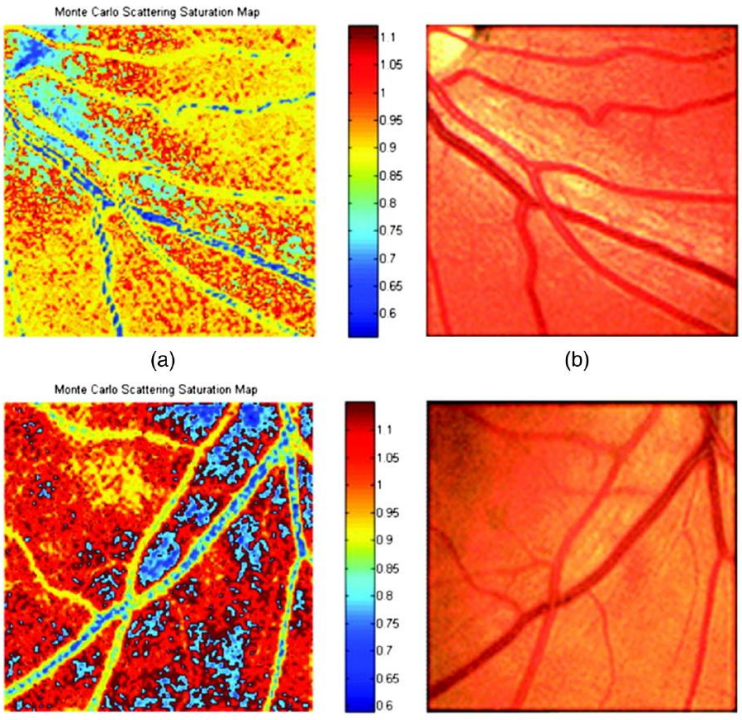


Fig. 2.16. Mapas de saturación de oxígeno. (a) Mapa de un varón de 29 años (b) Imagen normal (c) Mapa de un varón de 58 años (d) Imagen normal (Fuente: http://biomedicaloptics.spiedigitallibrary.org/)

A continuación, se realizará un breve estudio de las dos categorías anteriormente mencionadas. Conviene destacar que este estudio se centrará únicamente en los aspectos relacionados con la detección y ayuda a la extracción de tumores cancerígenos, puesto que, como se mencionó en el capítulo 1, este Proyecto Fin de Grado se inscribe dentro del proyecto europeo HELICoiD, cuyo objetivo principal es el desarrollo de un sistema no invasivo de ayuda a la detección y localización de cáncer cerebral durante una intervención quirúrgica. En consecuencia, HELICoiD supone una combinación de las dos categorías anteriormente mencionadas, puesto que, para poder asistir y guiar a un cirujano en la extracción del tumor, éste debe ser localizado en primer lugar. Esta asistencia es especialmente valorada por los cirujanos, puesto que, tradicionalmente, el éxito de este tipo de extirpaciones depende, en gran medida, de la pericia, habilidad y experiencia del cirujano para identificar la lesión y sus márgenes [56].

Por ello, la aplicación de esta tecnología como herramienta de ayuda visual supone una mejora sustancial en las intervenciones de extracción de tumores, ya que amplía la visión del cirujano a niveles moleculares o incluso celulares y, además, proporciona información que el ojo humano no puede captar. Este es el caso, por ejemplo, de la presencia de sangre; durante una operación, la existencia de sangre en los alrededores de la zona afectada es inevitable y, además, supone un gran contratiempo para el cirujano, que no puede observar con claridad el tejido y, por tanto, no puede delimitar con precisión la zona afectada de la sana. En este sentido, existen varios estudios, como [57], que han probado la bondad de estos sistemas de visualización mediante el análisis de tejidos sumergidos en una capa de sangre.

Además de la presencia de sangre en la zona a extirpar, otro punto crítico de las cirugías tumorales radica en, precisamente, la determinación de los márgenes entre tejido canceroso y sano [53]. Como se ha mencionado anteriormente, esta determinación depende, en gran medida, de la experiencia y pericia del cirujano, que es el que decide, en última instancia, qué zona se ha

de extirpar. Habitualmente, junto al tejido canceroso suele extirparse, indistintamente, tejido sano, puesto que delimitar los bordes de un tumor sin extirpar tejido sano es una tarea prácticamente imposible al ojo humano. Esta circunstancia en determinados tipos de cáncer -como el cerebral- es especialmente crítica, ya que esa extracción innecesaria puede agravar las secuelas que posteriormente puedan presentarse en el paciente y, consecuentemente, empeorar su calidad de vida. En estos casos, la aplicación de las imágenes hiperespectrales es especialmente útil, puesto que proporciona al cirujano los medios necesarios para, en primer lugar, retirar el menor tejido sano posible y, en segundo lugar, detectar y localizar pequeños tumores residuales que no hayan sido eliminados previamente, y que suponen una de las principales causas de la alta mortalidad asociada a la recaída en este tipo de enfermedades [58].

A este respecto, a lo largo de los últimos años se han llevado a cabo diversos estudios que, de forma genérica, suelen estar desarrollados en animales – generalmente, ratas o cerdos – y que consisten en el análisis de muestras tanto de tejido sano como de tejido tumoroso. En función de la naturaleza de estas muestras, se habla de estudios *ex-vivo* – cuando la muestra ha sido previamente extirpada en una biopsia – y estudios *in-vivo* – cuando el análisis se realiza directamente sobre el cuerpo del sujeto–.

Por ejemplo, en [58] se proporciona un estudio de 2007 sobre la localización de tumores residuales durante la cirugía. Este experimento se ha llevado a cabo con ratas de laboratorio a las que se les ha provocado cáncer de mama y que, posteriormente, han sido sometidas a una cirugía para la extirpación del tumor. Durante el procedimiento quirúrgico, se ha utilizado un sistema MHSI para la localización, en primer lugar, del tumor y, en segundo lugar, de los tumores residuales, que no han sido extraídos de forma deliberada.

Los resultados extraídos han sido contrastados con un análisis histopatológico, y se ha obtenido un 89% de sensibilidad (proporción de positivos reales que son identificados como tales) y un 94% de especificidad (proporción de negativos que son correctamente identificados, esto es, el porcentaje de tejido sano que es correctamente identificado como no cancerígeno) en la detección de tumores residuales.

La figura 2.17 proporciona un resumen gráfico del proceso desarrollado:

- En primer lugar, en la figura 2.17 (A) se muestra una fotografía normal y una imagen hiperespectral del tumor antes de comenzar la intervención. Como puede observarse en la imagen hiperespectral, el tumor aparece resaltado en tonalidades azuladas.
- En segundo lugar, la figura 2.17 (B) muestra, nuevamente, una fotografía normal y una imagen hiperespectral del tumor, pero esta vez el tumor está descubierto – esto es, se ha retirado el tejido que lo recubre-. Como puede observarse, el tumor ahora se aprecia de una forma mucho más nítida, mostrándose, en un azul más claro, el nodo principal de dicho tumor.
- En tercer lugar, en la figura 2.17 (C) se muestra el lecho tumoral una vez se ha extraído gran parte del tumor, entendiéndose por lecho tumoral el tejido sano sobre el que se desarrolla dicho tumor. En la imagen hiperespectral pueden observarse a la perfección los tumores residuales que, intencionadamente, no han sido extirpados.
- Por último, la figura 2.17 (D) muestra el lecho tumoral tras finalizar la extirpación del tumor. Como puede observarse en la imagen hiperespectral, este lecho está limpio, puesto que no queda ningún rastro de tejido canceroso, que debería mostrarse en azul.

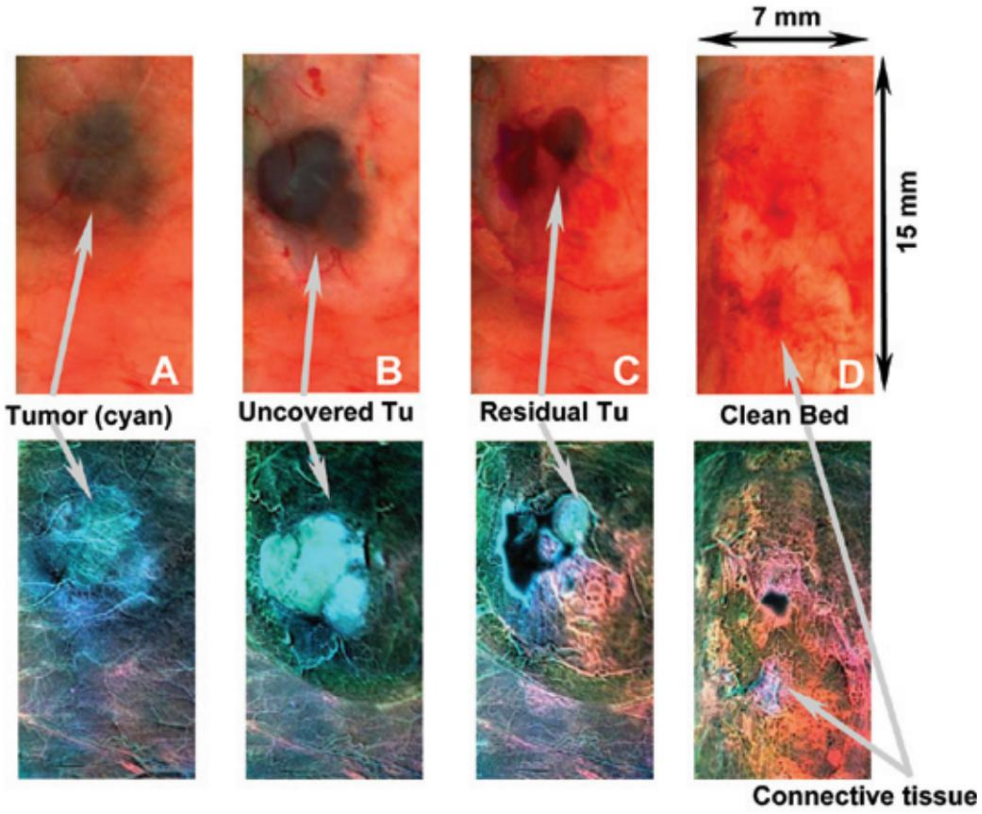


Fig. 2.17. Imágenes hiperespectrales del tumor y del lecho tumoral. Microfotografías en color real del tumor (A); del tumor expuesto (B); de los tumores residuales (C) y del lecho tumoral limpio (D) [58]

En [59] se proporciona otro ejemplo de detección de tumores mediante técnicas MHSI. En este caso, también se trata de un estudio *in-vivo* para localizar tumores en ratones, pero esta vez el tumor es de próstata. Al igual que en el estudio anterior, para este experimento se han utilizado ratones de laboratorio a los que se les ha provocado un cáncer de próstata humano y, posteriormente, han sido fotografiados por un sistema de imágenes hiperespectrales.

Los resultados del estudio se miden en sensibilidad y especificidad, obteniendo un 92.8 ± 2.0% de sensibilidad y un 96.9% ± 1.3% de especificidad. Esto supone un 7.2% de falsos negativos y un 3.1% de falsos positivos. Conviene destacar que este experimento se ha realizado para un total de once ratones de laboratorio.

En la figura 2.18 se puede observar uno de los resultados obtenidos durante este experimento. En la figura 2.18 (A) se muestra la imagen real del sujeto de prueba; en la misma, puede apreciarse una masa informe creciendo a un costado del roedor. Por su parte, en la figura 2.18 (B) se observa el mapa de abundancias obtenido para ese determinado sujeto, en la que el color verde identifica al tejido cancerígeno; como se puede comprobar, la masa anteriormente mencionada aparece marcada con color verde, tal y como se esperaba. Sin embargo, cabe destacar que también aparecen otras zonas remarcadas en verde, y que, a simple vista, no se podían identificar como tejidos tumorales. Estas zonas se localizan en el lomo del roedor y en la cabeza, lo que parece indicar que el cáncer se ha extendido, afectando a otras zonas.

En consecuencia, puede concluirse que, ante los resultados obtenidos, la viabilidad del método propuesto en este estudio para la detección de cáncer de próstata en ratones ha quedado suficientemente comprobada.

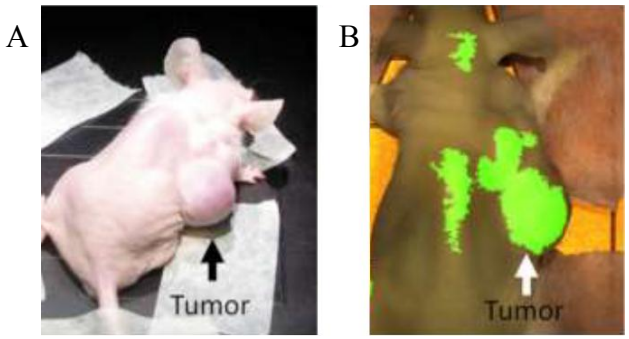


Fig. 2.18. (A) Ratón con cáncer de próstata (B) Resultado de la detección de cáncer tras aplicar un sistema MHSI [59]

Una vez demostrado que los resultados obtenidos en animales son favorables, el siguiente paso es comenzar a probar estos sistemas en humanos. Para ello, el primer paso es realizar pruebas sobre muestras *ex-vivo* - extirpadas mediante biopsia -, tal y como se realiza en [60]. En este estudio, se desarrolla un método de MHSI para detectar cáncer gástrico. Este sistema es evaluado con una serie de muestras obtenidas de 16 tumores gástricos, extirpados por endoscopia a 14 pacientes del *Yamaguchi University Hospital* (Japón). De cada uno de los tumores, se han tomado 10 muestras de tejido canceroso y otras 10 de tejido normal, lo que arroja un total de 320 muestras a analizar. De esas 320 muestras, la mitad se han destinado a *training* (calibración o entrenamiento) del sistema, y la otra mitad se ha utilizado como prueba de este método. Tras la calibración y prueba del sistema, los resultados obtenidos han sido [60]:

- *Sensibilidad*: Este estudio prueba 160 muestras, de las cuales 80 son cancerígenas. De estas 80 muestras, 63 son correctamente clasificadas, lo que supone un 78.8% de sensibilidad y un 21.2% de falsos negativos.
- *Especificidad*: De las 80 muestras de tejido sano, en este estudio se han detectado 74 como tal, lo que arroja una proporción del 92.5% de especificidad y un 7.5% de falsos positivos.
- *Precisión*: En total, se han analizado 160 muestras y, de ellas, se han clasificado correctamente 137. Esto supone un 85.6% de precisión.

En la figura 2.19 se proporciona un ejemplo de los resultados obtenidos para el análisis de uno de los tumores utilizados durante el estudio (nombrado Caso 1 en el mismo). A la izquierda, se proporciona la firma espectral de las 10 muestras de tejido canceroso (línea continua) y las 10 muestras de tejido sano (línea discontinua). Esta firma representa la llamada *reflectancia espectral corregida*, que se define en [60] como la reflectancia espectral de cada muestra menos la reflectancia espectral media de 5 muestras sanas. Esto hace que, como puede apreciarse en la figura, las firmas difieran sustancialmente a partir de cierto punto (en este caso, 580 nm aproximadamente), lo que permite la eficaz localización de las áreas cancerosas. A la derecha, se proporcionan varias imágenes obtenidas del mismo tumor al que pertenecen las firmas espectrales mostradas en la figura 2.19. Como puede observarse, la figura 2.19 (b) proporciona un análisis histopatológico de la muestra extraída, en las que las zonas rojas se corresponden con áreas tumorales, y en la figura 2.19 (d) se aporta el mapa de abundancias obtenido tras el procesado de la imagen hiperespectral para la banda de 726 nm que, como se puede observar en la figura 2.19 (izquierda), es una de las bandas donde más diferencia existe entre las firmas. Conviene destacar que la escala de colores de esta última se corresponde con la escala que aparece en las firmas espectrales; en consecuencia, los valores más cercanos a 0.2 se identifican con color rojo, y los más cercanos a -0.05, con color azul. Esto hace que, en la figura 2.18 (d), la localización de las zonas afectadas por el cáncer sea mucho más efectiva que en la (b).

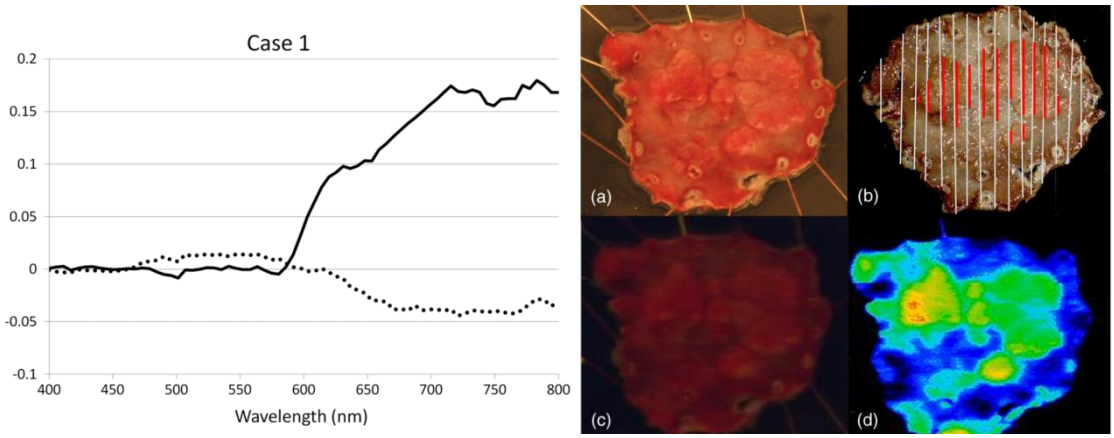


Fig. 2. 19. (Izquierda) Firma espectral de las muestras de uno de los tumores. La línea continua se corresponde con tejidos tumorales y la discontinua con tejido sano. (Derecha) Imágenes tomadas del tumor al que corresponden las firmas espectrales. (a) Imagen digital normal (b) Imagen normal dividida en secciones de 2 mm de separación. (c) Imagen hiperespectral para la banda de 726 nm antes del procesado (d) Imagen hiperespectral procesada [60].

Por último, para concluir este apartado se va a proporcionar un ejemplo de aplicación *in-vivo* de MHSI en humanos. Por ejemplo, en [61] se proporciona un estudio sobre detección de cáncer de lengua en humanos mediante la aplicación de un sistema MHSI combinado con filtros espectrales acústico – ópticos ajustables (AOTF, del inglés *Acousto – Optic Tunable Filter*). El motivo de esta combinación se basa en que, debido a la naturaleza de este tipo de cáncer, los sistemas MHSI tradicionales no arrojan resultados óptimos. Esto se debe a que tanto los movimientos reflejos de la lengua como la presencia de saliva se traducen en un ruido añadido que provoca una disminución sustancial de la precisión con la que se detectan los tumores. Los resultados obtenidos con este nuevo sistema son muy favorables: el conjunto MHSI–AOTF consigue una precisión de un 96.5%, con un porcentaje de falsos positivos y falsos negativos de, respectivamente, un 6.3% y un 8.7%, ambos sobre el total de píxeles analizados.

La figura 2.20 aporta un ejemplo del resultado obtenido por el sistema desarrollado en este estudio. A modo de comparativa, en la izquierda se proporciona el análisis histopatológico realizado por un experto, en el que aparecen demarcadas en verde las zonas cancerígenas; a la derecha se proporciona el resultado obtenido por el sistema MHSI – AOTF. Como puede apreciarse, los resultados son prácticamente idénticos, con la diferencia de que el sistema desarrollado señala otras zonas, muy pequeñas, en las que se detectan tejidos tumorales. Esto parece demostrar, además, que este sistema es capaz de detectar y localizar tumores residuales o de pequeño tamaño.

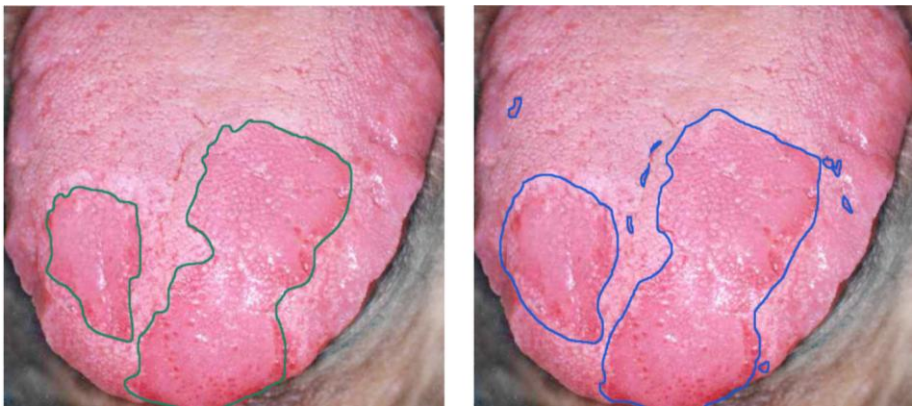


Fig. 2. 20. Región con tumor. Clasificación de experto (izquierda) y del sistema MHSI - AOTF (derecha) [61]

2.3. Imágenes hiperespectrales: velocidad de análisis

Para concluir con el estudio de las imágenes hiperespectrales, el último punto que queda por analizar es el del tiempo de procesado de una imagen hiperespectral. Esto es especialmente importante en este Proyecto Fin de Grado puesto que, como se mencionó en el capítulo 1, uno de los objetivos es la consecución, en la medida de lo posible, del análisis de una imagen hiperespectral en tiempo real. Por ello, un paso previo al desarrollo de la solución propuesta en este Proyecto es el análisis del estado del arte en este ámbito. Cabe destacar que el concepto de tiempo real que este Proyecto busca no es el tradicional de los decodificadores digitales de vídeo (entre 30 y 40 imágenes por segundo); en el contexto de las imágenes hiperespectrales, se podrá considerar que se ha alcanzado el tiempo real si se consigue procesar una imagen en el tiempo que tarda el sistema de adquisición en proporcionar la siguiente [62].

A continuación se proporciona un análisis del estado del arte en lo referente a la velocidad de análisis de imágenes hiperespectrales. No obstante, conviene destacar que, de forma genérica, la mayoría de la literatura consultada no aporta información cuantitativa sobre tiempos computacionales, por lo que sólo se han tenido en cuenta en esta sección aquellos estudios que, de una forma u otra, aportan mediciones de tiempos de computación. A este respecto, la tabla 2.1 recoge un resumen de algunos de los estudios encontrados, en los que los datos se han obtenido tras la ejecución de las distintas aplicaciones en sistemas mononúcleo. Como se puede observar, los resultados obtenidos no son concluyentes, puesto que los tiempos varían considerablemente. No obstante, estos resultados tampoco son comparables entre ellos, puesto que tanto las imágenes hiperespectrales empleadas como el entorno son muy dispares.

Ref.	Año	Aplicación	Tiempo	Entorno	Píxeles	Nº bandas
[63]	2003	Detección de golpes en manzanas	2 – 3 s	C++	320 x 240	80
[64]	2005	Corrección atmosférica	≈ 60 s	-	512 x 512	224
[58]	2007	Detección de tumores residuales	30 s	Matlab	1024 x 1528	34
[65]	2010	Detección de isquemia intestinal	< 10 s	Matlab	484	121
					240	157
[66]	2011	Detección de tumores	< 20 s	Matlab	-	-

Tabla 2.1. Comparativa de tiempos de computación para ejecución en serie

Lo que sí se ha observado durante este análisis es que una gran cantidad de estudios de los últimos años resaltan la necesidad de utilizar sistemas multinúcleo que proporcionen paralelismo al procesamiento de imágenes para la consecución del tiempo real. En este sentido, en los últimos años se han publicado varios estudios en los que la implementación de la cadena de procesado se realiza en *Grafics Processing Units* o GPUs, que son un tipo de arquitecturas hardware multinúcleo cuya principal característica es la gran cantidad de procesadores integrados [67-70]. Por ejemplo, en [67] se proporciona una comparativa del tiempo de ejecución de varios algoritmos de extracción de *endmembers* en distintas GPUs; en [68], se implementa una cadena de desmezclado en la que la elección de algoritmos se basa en su facilidad de paralelización y se ejecuta tanto en un sistema mononúcleo como en una GPU; en [69] se proporciona un estudio cuantitativo sobre la facilidad de paralelización de dos algoritmos de extracción de *endmembers*, analizando la evolución del tiempo computacional de cada uno de ellos a medida que aumenta el número de procesadores; y, por último, en [70] se proporciona un análisis comparativo de la aceleración conseguida en el procesamiento de una imagen hiperespectral real en una GPU respecto a un sistema mononúcleo.

En este sentido, la tesis doctoral de Sergio Sánchez Martínez [11] proporciona un amplio estudio sobre el tiempo computacional asociado al procesado de una imagen hiperespectral real, aportando una extensa comparativa de la diferencia de tiempo de procesado entre una ejecución secuencial y una ejecución en paralelo. Para ello, construye varias cadenas de procesamiento de imágenes hiperespectrales, combinando distintos algoritmos para cada una de las fases de la cadena y, a continuación, ejecuta cada una de ellas en tres plataformas distintas:

- En primer lugar, ejecuta las distintas cadenas en un único núcleo. Para ello, la plataforma utilizada es una CPU con un procesador Intel Core i7. Conviene destacar que, aunque este procesador es multinúcleo, en la ejecución sólo se utiliza uno de los disponibles, simulando así el comportamiento de un sistema mononúcleo.
- En segundo lugar, ejecuta las distintas cadenas en dos GPUs distintas, de tal forma que la ejecución se efectúa en paralelo. Las GPUs utilizadas en este apartado son la *NVidia™ Tesla C1060* y la *NVidia™ GTX 580*.
- Por último, además de ejecutar las cadenas en distintas plataformas, para cada una de ellas utiliza dos compiladores distintos - *gcc* e *icc* -, lo que permite observar el impacto de la compilación en el tiempo de ejecución.
- Cabe mencionar que, para la evaluación de las distintas cadenas de procesado, el autor utiliza dos imágenes hiperespectrales reales que han sido ampliamente utilizadas en la literatura de desmezclado espectral: la *AVIRIS Cuprite*[14] y la *AVIRIS World Trade Center (WTC)*[15]. Ambas fueron tomadas por el sensor de la NASA *AVIRIS*, la primera en 1995 sobre un paraje de Nevada y la segunda en 2001, cinco días después del ataque terrorista al *World Trade Center*.

En consecuencia, este estudio elabora un amplio análisis del impacto de la paralelización de las cadenas de procesado sobre el tiempo computacional, permitiendo la elección de la cadena óptima en este aspecto y, justificando, además, la necesidad de utilización de plataformas multinúcleo para la consecución del tiempo real. Asimismo, también proporciona resultados suficientes para determinar el tipo de compilador que proporciona resultados óptimos para el tiempo de ejecución. En concreto, los algoritmos que el autor utiliza para la implementación de las distintas cadenas son: VD y HYSIME para la fase de estimación del número de *endmembers*; PCA y SPCA para la fase de reducción dimensional; N-FINDR para la fase de extracción de *endmembers*; y, por último, UCLS y NCLS para la fase de estimación de abundancias. Esto arroja un total de ocho posibles combinaciones, que se muestran en la tabla 2.2.

Cadena	FASE 1		FASE 2		FASE 3	FASE 4	
	VD	*HYSIME*	*PCA*	*SPCA*	*N-FINDR*	*UCLS*	*NCLS*
C1	X		X		X	X	
C2	X		X		X		X
C3	X			X	X	X	
C4	X			X	X		X
C5		X	X		X	X	
C6		X	X		X		X
C7		X		X	X	X	
C8		X		X	X		X

Tabla 2.2. Cadenas de procesamiento

[14] http://aviris.jpl.nasa.gov/data/free_data.html
[15] http://aviris.jpl.nasa.gov/links/

A continuación se aporta, a modo ilustrativo, un resumen de los resultados obtenidos en este análisis. Para ello, se van a replicar los resultados extraídos por el autor de la tesis para la imagen *Cuprite*. En particular, los resultados que se mostrarán aquí son los obtenidos para el mejor y el peor caso. El mejor caso se obtiene con las cadenas 1 y 3, en las que el tiempo de ejecución en serie es ligeramente inferior a los 12.4 segundos; en la tabla 2.3 se aportan los resultados obtenidos para la cadena 1, que son ligeramente mejores a los de la cadena 3, aunque la diferencia es de apenas unos milisegundos. Por su parte, el peor caso se obtiene con las cadenas 6 y 8, que, al igual que en el mejor caso, apenas se diferencian entre ellas en unos cuantos milisegundos. Como los resultados de la cadena 8 son ligeramente peores, éstos son los que se aportan en la tabla 2.4.

	LOAD	VD	PCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.186	5.555	5.401	0.883	0.313	0.008	12.346	
Tesla gcc	0.202	0.411	0.147	0.112	0.054	0.008	0.934	13.219
GTX gcc	0.204	0.423	0.137	0.107	0.045	0.010	0.926	13.331
Serie icc	0.170	3.940	2.253	0.847	0.251	0.008	7.470	
Tesla icc	0.183	0.420	0.149	0.115	0.054	0.012	0.933	8.007
GTX icc	0.194	0.441	0.132	0.094	0.041	0.012	0.914	8.174

Tabla 2.3. Tiempo de procesamiento de las distintas etapas de C1 [11]

	LOAD	HYSIME	SPCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.179	28.282	5.428	0.600	77.685	0.007	112.181	
Tesla gcc	0.197	1.853	0.169	0.083	1.075	0.007	3.383	33.161
GTX gcc	0.201	1.655	0.145	0.069	0.859	0.007	2.936	38.207
Serie icc	0.164	20.100	2.222	0.526	61.435	0.007	84.453	
Tesla icc	0.183	1.842	0.171	0.085	1.075	0.007	3.363	25.116
GTX icc	0.183	1.664	0.143	0.069	0.859	0.007	2.926	28.867

Tabla 2.4. Tiempo de procesamiento de las distintas etapas de C8 [11]

De los resultados recogidos en estas dos tablas se pueden extraer una serie de conclusiones:

- En primer lugar, puede observarse que, de forma general, la utilización del compilador *icc* arroja mejores resultados que la utilización del compilador *gcc*; en concreto, para el mejor caso, el *icc* supone un ahorro en tiempo de ejecución de un 39.49%.
- En segundo lugar, se puede observar que, en ambos casos, la disminución de tiempo conseguida con la utilización de las GPUs es sustancial, llegando a obtenerse una tasa de aceleración de 40.
- Por último, de los resultados obtenidos también se puede deducir que, en general, el algoritmo que más retarda al sistema es el NCLS, que implica un tiempo de ejecución 250 veces superior al UCLS.

Para concluir este apartado, en la figura 2.21 se proporciona una comparativa de los distintos tiempos de procesamiento de ambas GPUs para cada una de las cadenas anteriormente explicadas. Además, en el gráfico aparece una línea roja que marca el límite del tiempo real que, en esta aplicación, viene dado por el tiempo que tardó el sensor *AVIRIS* en captar esta imagen. Como puede observarse, existen dos cadenas (1 y 3) en las que se alcanzaría el tiempo real en ambas GPUs, mientras que, en otras dos (2 y 4) sólo alcanzaría el tiempo real una de las GPUs.

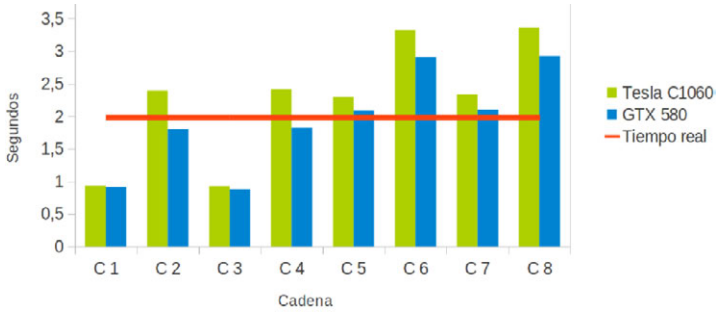


Fig. 2.21. Comparativa de las cadenas [11]

2.4. Plataformas multinúcleo

El análisis elaborado en el apartado anterior permite extraer varias conclusiones, siendo la más importante la necesidad de paralelización de la cadena de desmezclado cuando el tiempo de ejecución supone un punto crítico en la aplicación concreta de esta tecnología. Como se ha visto en [11], las únicas ocasiones en las que se ha alcanzado el tiempo real han sido aquellas en las que la cadena de desmezclado se ejecutaba en un sistema multinúcleo. Teniendo en cuenta que, como se ha mencionado anteriormente, la aplicación a la que está orientado este Proyecto Fin de Grado es la detección y localización de tumores cerebrales durante una cirugía, resulta evidente que el tiempo de procesamiento de las imágenes capturadas es uno de los puntos críticos de todo el proyecto. Como se mencionó en el capítulo 1, uno de los objetivos es la consecución del tiempo real en el procesamiento, entendiéndolo como el tiempo que emplea el sensor en proporcionar una imagen hiperespectral, de tal forma que, cuando se proporcione la siguiente, el sistema pueda empezar a analizarla automáticamente, sin añadir retardos adicionales.

En consecuencia, una vez justificada la necesidad de paralelización, el siguiente paso es la elección de una plataforma multinúcleo que proporcione los recursos necesarios para alcanzar este objetivo. Para ello, se necesita realizar un estudio comparativo de las plataformas existentes en el mercado. Este estudio aparece detallado en profundidad en el Proyecto Fin de Grado de Daniel Madroñal Quintín [1], por lo que en este apartado sólo se recogerá un resumen del mismo, que puede observarse en la tabla 2.5, donde se evalúan distintos tipos de plataformas. Obviamente, las primeras son las GPUs, puesto que, como se ha visto en el apartado anterior, su eficiencia en este tipo de aplicaciones está suficientemente probada. No obstante, existen otras plataformas en el mercado que pueden aportar los recursos necesarios para la ejecución de una cadena de desmezclado en tiempo real, por lo que también se han tenido en cuenta en dicho estudio. En concreto, estas alternativas son las siguientes: los *Intel Xeon Phi Coprocessors,* la *MPPA256* de *Kalray* y los *HyperX Processors* de *Coherent Logix*. En la tabla 2.5 se recogen las principales características de algunas de las GPUs más potentes del mercado – la *NVidia™ Tesla K40* y las *FirePro* de *AMD* –, así como las de las plataformas alternativas ya mencionadas. De esta tabla se pueden extraer las siguientes conclusiones:

- De forma general, resulta evidente que las mejores alternativas en cuanto a número de procesadores y velocidad son las GPUs. Sin embargo, el gran problema que éstas presentan es que, en general, utilizan lenguajes de programación muy específicos, como *CUDA*, lo que implica la necesidad de un experto en el lenguaje que se encargue de la programación de la misma. Esto resulta un inconveniente en proyectos conjuntos, como HELICoiD, en los que distintos grupos necesitan hacer aportaciones a la aplicación.
- Otras plataformas que pueden resultar interesantes son la *MPPA256* y la *HyperX hx3100* que, aunque no pueden competir con las GPUs en número de núcleos y velocidad, el consumo que presentan es casi 50 veces mejor.

En conclusión, aunque parece evidente que las GPUs siguen siendo las opciones más potentes en cuanto a velocidad de procesamiento, conviene tener en cuenta otras alternativas que, si bien presentan una capacidad de cómputo menor, aportan otras características provechosas, como una disminución sustancial en el consumo y una mayor facilidad de programación. Por todo ello, para decantarse por una opción u otra habría que analizar en detalle las especificaciones particulares de la aplicación para la que se va a utilizar la plataforma –en este caso, HELICoiD-, estudiando qué características priman sobre las demás. Además, no se debe olvidar que, para elegir una alternativa, otro factor a tener en cuenta sería el precio de cada plataforma, que no se ha contemplado en este estudio porque no se ha encontrado suficiente información al respecto, pero que, finalmente, puede ser el determinante de la elección.

Plataforma	Empresa	Procesadores	Velocidad	Consumo	Rendimiento	Memoria	Lenguaje
Tesla K40	NVIDIA	>2800	· 4.29 TF – 32 bits · 1.43 TF – 64 bits	235 W	· 18.25 GF/W – 32 bits · 6.08 GF/W – 64 bits	12 GB – 288 GB/s	CUDA C/C++ OpenCL
FirePro W9100	AMD/ATI	2816	· 5.24 TF – 32 bits · 2.62 TF – 64 bits	275 W	· 19.05 GF/W – 32 bits · 9.53 GF/W – 64 bits	6 GB – 480 GB/s	OpenCL
FirePro S10000	AMD/ATI	2x1792	· 5.91 TF – 32 bits · 1.48 TF – 64 bits	275 W	· 21.5 GF/W – 32 bits · 5.38 GF/W – 64 bits	6 GB – 480 GB/s	OpenCL
Intel Xeon	Intel	58-60-61	1003-1011-1208 GF – 64 bits	225-300 W	4-4.5 GF/W	6-16 GB – 240-352 GB/s	C/C++ Fortran
MPPA 256	Kalray	256	230 GF	5 W	50 GF/W	-	C/C++ OpenCL
HyperX hx3100	Coherent Logix	100	25 GF – 32 bits	2.5 W	10 GF/W	-	C

Tabla 2.5. Resumen de prestaciones de plataformas multinúcleo [1]

2.5. RVC – CAL

En apartados anteriores se ha justificado la necesidad de paralelización de la cadena de desmezclado de imágenes hiperespectrales y, en consecuencia, la necesidad de utilizar una plataforma multinúcleo; sin embargo, todavía no se ha explicado cómo abordar esa paralelización. Parece evidente que, para reducir el tiempo de ejecución de la cadena, la metodología a aplicar es: en primer lugar, buscar los puntos críticos o cuellos de botella de la misma; en segundo lugar, buscar posibles soluciones a esos cuellos de botella; y, en tercer lugar, localizar las partes de la cadena que pueden ejecutarse en paralelo, de tal forma que se asigne cada una de ellas a un procesador distinto. Sin embargo, aplicar esta metodología en lenguajes convencionales, como C, C++ o Java, es una tarea compleja, puesto que el paralelismo de la aplicación no se observa de forma explícita.

En este sentido, existe un tipo de lenguaje que podría ser muy útil para la extracción de este paralelismo, que es el lenguaje de flujo de datos. De forma genérica, este lenguaje se caracteriza por estar compuesto de un conjunto de elementos o nodos que se comunican entre sí, intercambiándose paquetes de datos, y que realizan acciones con los datos que reciben.

Por ejemplo, un tipo de lenguaje basado en flujo de datos es RVC – CAL (de sus siglas en inglés, *Reconfigurable Video Coding – CAL Actor Language*), que está enfocado al desarrollo de aplicaciones divididas en bloques con funcionalidades independientes y que se comunican entre ellos. Para la realización de esta división en bloques, RVC – CAL presenta una estructura básica compuesta por tres elementos: actores, acciones y *networks* o redes.

- El actor es la unidad funcional de este lenguaje. Continuando con la terminología anterior, los actores se corresponden con lo que se ha denominado nodo o bloque, y se caracterizan por presentar una serie de puertos de entrada y salida y desarrollar una o varias funciones.
- Dentro de un actor, cada una de las funcionalidades que desarrolla se implementa en una acción distinta. Por tanto, las acciones pueden definirse como cada una de las distintas funciones que implementa un determinado actor.
- En general, una red o *network* es un conjunto de actores conectados entre sí, que presenta unos puertos de entrada y unos puertos de salida. De forma genérica, una *network* se representa como un diagrama de bloques, en el que cada bloque se corresponde con un actor, y en el que los arcos que unen unos bloques con otros representan los distintos intercambios de datos – o *tokens* –. Un ejemplo de esto puede observarse en la figura 2.22, que muestra una *network* compuesta por seis actores. Cabe destacar que los bloques que conforman las *networks* no tienen por qué ser siempre actores, sino que pueden ser otras *networks*.

La estructura de este lenguaje hace que la problemática de la extracción del paralelismo se simplifique considerablemente, puesto que, como se puede observar en la figura anterior, se puede observar a simple vista qué actores pueden ejecutarse de forma simultánea – como, por ejemplo, los actores llamados *Algo_IntraPred_LU* y *Algo_Split_16x16* –. Esta característica hace que el lenguaje esté especialmente orientado a aplicaciones multimedia. Por ejemplo, el grupo MPEG (de sus siglas en inglés, *Moving Pictures Experts Group*) está desarrollando un nuevo estándar de codificadores de vídeo reconfigurables basado en este lenguaje, de tal forma que se pueda modificar, de forma dinámica, alguno de los actores que componen el decodificador sin afectar al resto [71].

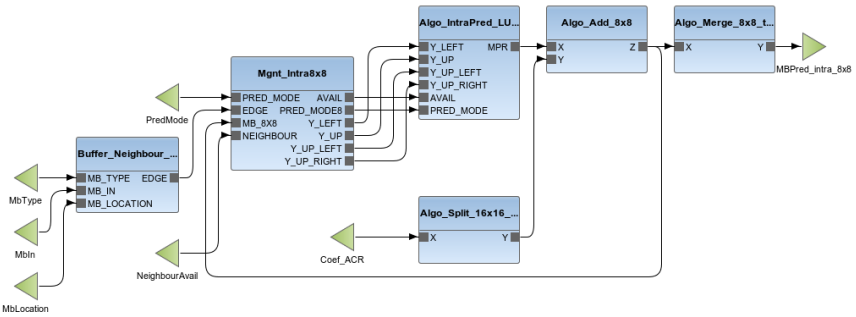


Fig. 2.22. Ejemplo de network (Fuente: http://orcc.sourceforge.net/)

Por último, para concluir este apartado se van a resumir las ventajas e inconvenientes que presenta la utilización de este lenguaje para la implementación de aplicaciones multimedia, en general, y de aplicaciones de procesado de imágenes hiperespectrales, en particular.

Ventajas

- Sin duda, la principal ventaja de este lenguaje es la simplificación en la extracción del paralelismo de las aplicaciones, que facilita la distribución de actores en distintos procesadores. De hecho, se ha desarrollado un *plugin* de este lenguaje para Eclipse que permite realizar directamente dicha distribución.
- Además, el compilador utilizado por este lenguaje (denominado ORCC, de sus siglas en inglés *Open RVC – CAL Compiler*) permite generar el código de la aplicación desarrollada en distintos lenguajes tradicionales, como C, C++ o Java. Esto supone una importante ventaja, puesto que garantiza la interoperabilidad entre distintas plataformas.
- Otra de las ventajas de este lenguaje es que, como se ha mencionado anteriormente, la implementación de la aplicación en distintos actores permite la modificación rápida y sencilla de secciones de código, puesto que los cambios en un actor no afectan al resto. Esto es especialmente útil en el procesado de imágenes hiperespectrales, ya que se puede implementar cada fase de la cadena en un actor y, así, se facilita la modificación de, por ejemplo, los algoritmos implementados en cada etapa.
- Por último, otra ventaja que se ha de mencionar es la posibilidad de importar librerías codificadas en lenguajes compatibles con el generado, de tal forma que se puede ampliar de forma externa el conjunto de funcionalidades de RVC – CAL.

Inconvenientes

- Una de las principales desventajas es la escasez tanto de librerías propias como de recursos del lenguaje; al estar muy orientado a la codificación y decodificación de vídeo, este lenguaje presenta ciertas carencias respecto a los lenguajes tradicionales.
- Otra desventaja a tener en cuenta es que este lenguaje no genera ejecutables, sino que genera código en otro lenguaje – como C – a partir de la descripción implementada en RVC - CAL. Por ello, para obtener el ejecutable se debe compilar dicho código autogenerado con un compilador tradicional, como *gcc*.
- Al ser un lenguaje en desarrollo, la frecuencia de actualizaciones del mismo es alta, pudiendo causar ciertas incompatibilidades con las versiones anteriores. Asimismo, la información disponible acerca del lenguaje es escasa y poco accesible.

En conclusión, este lenguaje presenta un gran potencial para la paralelización de aplicaciones y, en consecuencia, para la disminución del tiempo de ejecución de las mismas. Por ello, se ha tomado la decisión de utilizar este lenguaje en el desarrollo de este Proyecto. Si se desea conocer más detalles de este lenguaje, en [1] se ha realizado un análisis pormenorizado. Asimismo, en [72] se proporciona el manual de usuario del mismo.

2.6. PAPI

Como se ha mencionado anteriormente, una de las acciones a realizar a la hora de reducir el tiempo de ejecución de la cadena de procesado de imágenes hiperespectrales es la detección y localización de los posibles cuellos de botella. En este sentido, existe una nueva herramienta de análisis de rendimiento denominada PAPI (de sus siglas en inglés, *Performance Application Programming Interface*) que proporciona, precisamente, esta funcionalidad.

Para finalizar el presente capítulo, se va a realizar un breve análisis de esta nueva herramienta. Cabe destacar que el análisis completo se ha desarrollado en el Proyecto Fin de Grado de Daniel Madroñal Quintín [1].

PAPI es una librería que proporciona las funciones necesarias para la monitorización de una serie de registros hardware presentes en la mayoría de los procesadores modernos, que proporcionan información que puede ser utilizada para aplicaciones como: optimización de la compilación, testeo de aplicaciones, bancos de pruebas, monitorización y modelado del rendimiento. En concreto, en [73] se proporciona la lista completa de los registros a los que PAPI puede acceder, organizados por funcionalidad.

Una de las principales características de PAPI es su interoperabilidad. Esto se debe, principalmente, a que las funciones de esta librería están desarrolladas de tal forma que pueden ser utilizadas en prácticamente cualquier tipo de plataforma, ya que, como se ha mencionado anteriormente, los registros examinados por PAPI son comunes a la mayoría de las arquitecturas hardware actuales. Esta característica supone una enorme ventaja para HELICoiD, en general, y para este Proyecto Fin de Grado, en particular, puesto que garantiza el correcto funcionamiento de PAPI aunque la plataforma aún no haya sido elegida.

Por último, otra de las ventajas más importantes de PAPI es que permite la monitorización de varios hilos simultáneamente. Esto, aplicado a un sistema multinúcleo, permite monitorizar el rendimiento de cada procesador por separado, proporcionando así información fiable sobre los puntos críticos del sistema debidos a la paralelización de la aplicación.

A modo de ejemplo, en la figura 2.23 se observa un ejemplo de monitorización en el que esta herramienta localiza un cuello de botella debido a accesos a memoria caché fallidos.

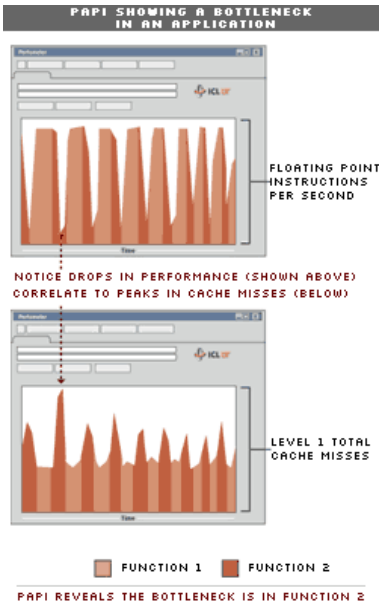


Fig. 2.23. Detección de un cuello de botella usando PAPI
(Fuente: http://icl.cs.utk.edu/papi/overview/index.html)

CAPÍTULO 3. DESCRIPCIÓN DE LA SOLUCIÓN

En el presente capítulo se explicará de forma exhaustiva la implementación de la librería mencionada en el capítulo 1, así como la utilización de la misma para el desarrollo de una cadena completa de análisis de imágenes hiperespectrales en el lenguaje RVC - CAL. Este capítulo se divide en tres apartados: en primer lugar, se justificará la solución elegida, explicando las razones que han hecho que se decida utilizar el lenguaje RVC - CAL; después, se estudiarán los algoritmos de cada fase de la cadena y se elegirá uno de ellos para cada etapa; por último, se explicará el procedimiento de desarrollo, tanto de la librería como de la cadena.

3.1. Justificación de la solución elegida

En este apartado, se justificará la necesidad de implementar una librería de análisis de imágenes hiperespectrales que, además, esté construida en un lenguaje de flujo de datos como RVC - CAL. El capítulo anterior concluyó con la justificación de la necesidad de utilizar una plataforma multinúcleo cuando el tiempo de ejecución supone un factor crítico para la aplicación que se vaya a implementar. Como también se vio en capítulos anteriores, el trabajo que se va a explicar en el presente documento está enmarcado dentro del contexto de HELICoiD, lo que hace que la consecución del tiempo real sea uno de los objetivos a perseguir. Tras comprobar la necesidad de utilización de plataformas multinúcleo, el capítulo anterior introducía los lenguajes basados en flujo de datos y, en concreto, RVC - CAL. La aplicación de este tipo de lenguajes en plataformas multinúcleo era muy deseable, puesto que sus características permitían una explotación eficiente de los recursos de las mismas.

A continuación se recogen las ventajas y desventajas de este lenguaje, resumiendo las mencionadas en el capítulo 2 y detallando otras específicas de la aplicación a desarrollar.

Las ventajas que presenta este lenguaje son las siguientes:

- Generación automática de código en distintos lenguajes tradicionales. Gracias al uso del compilador ORCC, la descripción de la aplicación realizada en RVC - CAL puede traducirse automáticamente a lenguajes como C, C++ o Java. Esto supone una gran ventaja, puesto que independiza el uso de RVC - CAL del lenguaje en el que se programe una determinada plataforma, garantizando así la interoperabilidad entre ellas.
- Estructuración de la aplicación en nodos o *actores*. Como se vio en el capítulo 2, la principal ventaja de un lenguaje de flujo de datos como RVC - CAL es la distribución del código en unidades funcionales denominadas actores, que son independientes entre sí y pueden procesarse de forma simultánea. Esto permite, en primer lugar, la extracción del paralelismo inherente a la aplicación y, en segundo lugar, la rápida modificación del código; gracias a ello, si, por ejemplo, se identifica cada actor con una de las fases de la cadena explicadas en el capítulo 2, el algoritmo implementado en cada una de las fases puede modificarse o sustituirse por otro sin afectar al resto de fases de la cadena.
- Importación de librerías externas al lenguaje. Por último, otra de las principales ventajas de este lenguaje es que permite importar librerías externas - esto es, codificadas en otro lenguaje distinto a RVC - CAL -, siempre que estén implementadas en un lenguaje generable por el compilador ORCC. Como se verá a lo largo de este capítulo, esto proporciona una enorme ventaja, puesto que permite la posibilidad de implementar la librería desarrollada en este Proyecto en un lenguaje como C e importarla en RVC - CAL, de tal forma que la cadena completa se construya, desde el *plugin* de Eclipse, simplemente llamando a funciones pertenecientes a dicha librería.

Por su parte, las desventajas que presenta RVC - CAL para esta aplicación son las siguientes:

- Escasez tanto de librerías como de recursos propios del lenguaje. Como se mencionó en el capítulo 2, uno de los principales inconvenientes que presenta este lenguaje es que tanto las librerías como los recursos propios del lenguaje son bastante limitados. En el ámbito de este Proyecto, esto supone dos problemas inmediatos:
    - En primer lugar, la escasez de librerías propias como, por ejemplo, librerías con funciones matemáticas hace que el desarrollo de los algoritmos de la cadena de desmezclado espectral sean prácticamente imposibles de implementar únicamente en RVC - CAL, puesto que, como se vio en el capítulo 2, las imágenes hiperespectrales se expresan como una matriz bidimensional en la que la primera dimensión se corresponde con el número de píxeles y la segunda, con el número de bandas espectrales.
    - En segundo lugar, la escasez de recursos propios como, por ejemplo, determinados tipos de datos, hace que, como se verá más adelante, se dificulte la comunicación entre actores.
- Necesidad de dos compiladores. El hecho de que RVC - CAL no genere la aplicación en sí, sino que genere el código asociado a la descripción realizada en el mismo, hace que sea necesario un compilador tradicional, como *gcc*, además del citado compilador ORCC.

A la vista de las ventajas e inconvenientes de este lenguaje, se puede deducir que, si bien es muy deseable su utilización para el desarrollo de este Proyecto, lo cierto es que la implementación de la librería directamente en RVC - CAL no es viable, puesto que se necesitará una gran cantidad de operaciones con matrices y dicho lenguaje no dispone de ellas. Por tanto, parece quedar demostrado que, en consecuencia, esta librería ha de ser codificada en un lenguaje más potente en cuanto a operaciones matemáticas, de tal forma que se importe a RVC - CAL y se utilice para la implementación de la cadena de desmezclado. Cabe destacar que, tras discutir este problema con Eduardo Juárez - tutor de este Proyecto - y Daniel Madroñal, se ha decidido seguir esta metodología, implementando la librería en C y utilizándola en RVC - CAL. Así, se proporcionará una librería de procesado de imágenes hiperespectrales, que complementará al lenguaje y que, en un futuro, puede ser integrada en el propio RVC - CAL.

3.2. Contenido de la librería

Una vez justificada la necesidad de una librería de procesado de imágenes hiperespectrales, para empezar a implementarla es necesario estudiar previamente los algoritmos que se pueden utilizar para construir las fases mencionadas en el capítulo anterior. En consecuencia, en este apartado se realizará dicho estudio, describiendo cada algoritmo y analizando características como la complejidad, la velocidad de ejecución y las diferencias entre ellos. Tras esto, en cada fase se elegirá un algoritmo, que será estudiado en profundidad para extraer las funciones que lo forman y que, posteriormente, será implementado en RVC - CAL. Si la librería se ha construido correctamente, cuando esté completa se deberían poder implementar todos los algoritmos como una sucesión de llamadas a funciones pertenecientes a la misma, simplificando y minimizando el tiempo de desarrollo de cada algoritmo. Por tanto, la implementación de los algoritmos en RVC - CAL tiene una doble finalidad: por un lado, comprobar la bondad de los algoritmos implementados y, por otro, proporcionar la información necesaria para ampliar y mejorar la librería, ya que al desarrollar dichos algoritmos se pueden detectar tanto funciones que faltan como funciones que son comunes a varios algoritmos.

Precisamente, ésta última funcionalidad es una de las más importantes, puesto que el hecho de que para distintos algoritmos existan funciones comunes justifica la creación de la librería. Si, por el contrario, las funciones extraídas de cada algoritmo fuesen específicas para el mismo y no fueran utilizadas fuera de ese entorno, la creación de una librería no sería viable, puesto que se necesitarían estudiar todos y cada uno de los algoritmos existentes hasta el momento para extraer las funciones que los componen. En consecuencia, para realizar esta comprobación se ha decidido dividir la construcción de la librería en dos partes independientes; una de estas partes se desarrollará en este documento, mientras que la otra será la desarrollada por Daniel Madroñal [1]. Tras haber estudiado el proceso de análisis de una imagen hiperespectral y haber observado que éste se divide en cuatro fases, lo más natural ha sido realizar una división por etapas, de tal forma que cada alumno estudie una de las fases obligatorias y una de las fases opcionales. En concreto, en este Proyecto se estudiarán las fases de estimación del número de *endmembers* y estimación de abundancias, mientras que en el Proyecto de Daniel Madroñal se explicarán las fases de reducción dimensional y extracción de *endmembers*.

Tras la finalización del estudio por parte de ambos alumnos, se procederá a la puesta en común de las funciones extraídas en dichos proyectos para comprobar si ambas partes comparten funciones. Si la respuesta es afirmativa, quedará justificada la necesidad de desarrollar una librería de procesado de imágenes hiperespectrales; si, por el contrario, la respuesta es negativa, se plantearán otras alternativas.

Estimación del número de *endmembers*

Como ya se mencionó en el capítulo 2 del presente documento, el objetivo de esta fase radica en la obtención del número óptimo de *endmembers* - o firmas espectrales distintas - presentes en la imagen original. Por tanto, los algoritmos que componen esta fase presentan la misma interfaz de entrada y salida: inicialmente, parten de la imagen original y, a la salida, proporcionan el número óptimo de *endmembers* presentes en la misma. Los principales algoritmos desarrollados para implementar esta fase son VD (*Virtual Dimensionality*) y HYSIME (*HYperspectral Signal Identification with Minimum Error*), que se estudian a continuación.

El algoritmo VD es definido por sus desarrolladores como "el número mínimo de fuentes de señal distintas que caracterizan a los datos hiperespectrales desde la perspectiva de la detección y clasificación de objetivos" [74]. Este algoritmo se basa en la premisa de que, si un determinado *endmember* existe en la imagen hiperespectral, dicho *endmember* ha de presentar un pico característico en una determinada banda espectral, facilitando así su localización e identificación. En concreto, el funcionamiento básico de este algoritmo es el siguiente:

- En primer lugar, a partir de la imagen hiperespectral original de M píxeles por N bandas, se extraen las matrices de correlación (R_{NxN}) y de covarianza (K_{NxN}).
- En segundo lugar, se calculan los autovalores para cada banda de dichas matrices. Estos autovalores se denotan como λ'_i (para R_{NxN}) y λ_i (para K_{NxN}), en ambos casos con *i* tomando valores entre 1 y M.
- En tercer lugar, se realizan dos comprobaciones:

$$H_0 = \lambda'_i - \lambda_i = 0 \quad (3.1)$$

$$H_1 = \lambda'_i - \lambda_i > 0 \quad (3.2)$$

La hipótesis 3.1 prueba la ausencia de un determinado *endmember* en la i-ésima banda, mientras que la hipótesis 3.2 prueba, por el contrario, que dicho *endmember* sí está presente en dicha banda.

Para realizar esta comprobación, los autores se sirven de un método denominado *detector de Neyman - Pearson*, que ya fue utilizado satisfactoriamente para determinar el número de *endmembers* en imágenes obtenidas por el sensor AVIRIS, que ya ha sido mencionado en este documento. Este detector mide el número de veces que falla la hipótesis H_0 - o, lo que es lo mismo, cuántas veces se acierta la hipótesis H_1 - para cada banda existente en la imagen original. En consecuencia, el número de veces que dicha hipótesis falla se corresponde con el número de *endmembers* presentes en la imagen. Cabe destacar que este método utiliza otro parámetro de entrada, además de la mencionada imagen. Dicho parámetro es un valor de probabilidad de falsa alarma, P_F, que fija la sensibilidad del método [11, 74].

Analizando este algoritmo desde el punto de vista de la librería a desarrollar, las funciones que pueden extraerse son, entre otras, las siguientes:

- Obtención de la matriz de covarianza
- Obtención de la matriz de correlación
- Obtención de los autovalores de una determinada matriz

Por su parte, el algoritmo HYSIME se encuentra dividido, a su vez, en dos etapas muy diferenciadas: en la primera etapa se realiza una estimación del ruido presente en la imagen, mientras que en la segunda etapa es donde se calcula el número de *endmembers* [11, 13]. En consecuencia, la primera etapa de este algoritmo arroja como resultado una matriz de las mismas dimensiones que la matriz contenedora de la imagen hiperespectral real - es decir, M píxeles por N bandas -, que contiene la estimación de ruido presente en dicha imagen para cada píxel. En concreto, para calcular esta matriz se aplica una aproximación basada en regresión múltiple - o, en inglés, *multiple regression theory-based approach* -, que se basa en el hecho de que las bandas espectrales contiguas presentan un alto grado de correlación [13].

En la figura 3.1 se proporciona, a modo ilustrativo, el algoritmo que se emplea para implementar esta fase; como puede observarse, todas las operaciones son operaciones aritméticas (sumar y restar números) u operaciones con matrices (transponer una matriz, multiplicar dos matrices, invertir una matriz, etc).

Algorithm 1: Noise estimation
1) INPUT $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]$
2) $\mathbf{Z} := \mathbf{Y}^{\mathrm{T}}$, $\widehat{\mathbf{R}} := (\mathbf{Z}^{\mathrm{T}}\mathbf{Z})$;
3) $\mathbf{R}' := \widehat{\mathbf{R}}^{-1}$;
4) for $i := 1$ to L do
5) $\widehat{\boldsymbol{\beta}}_i := ([\mathbf{R}']_{\partial_i,\partial_i} - [\mathbf{R}']_{\partial_i,i}[\mathbf{R}']_{i,\partial_i}/[\mathbf{R}']_{i,i})[\widehat{\mathbf{R}}]_{\partial_i,i}$;
{Note that $\partial_i = 1, \ldots i-1, i+1, \ldots L$}
6) $\widehat{\boldsymbol{\xi}}_i := \mathbf{z}_i - \mathbf{Z}_{\partial_i}\widehat{\boldsymbol{\beta}}_i$;
7) end for
8) OUTPUT $\widehat{\boldsymbol{\xi}}$;
{$\widehat{\boldsymbol{\xi}}$ is an $N \times L$ matrix with the estimated noise}

Fig. 3.1. Algoritmo de estimación de ruido [13]

Respecto a la segunda etapa de este algoritmo, ésta se encarga de la estimación del subespacio en el que se inscriben los datos hiperespectrales, y comienza con la extracción de las matrices de correlación de la señal (R_x) y del ruido (R_n). Tras ello, aplicando el criterio del mínimo error cuadrático medio, se extrae el subconjunto de autovectores que mejor representa dicho subespacio. Al igual que en la primera fase, en ésta también se aporta, a modo ilustrativo, el algoritmo completo que se sigue para implementar esta etapa (ver figura 3.2).

Como se puede observar en dicha figura, las operaciones realizadas en este algoritmo también son, en su mayoría, operaciones aritméticas y de manejo de matrices: sumas, restas, divisiones, cálculos de matrices transpuestas, multiplicaciones de matrices, etc.

Algorithm 2: HySime

1) INPUT $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]$, $\widehat{\mathbf{R}}_y \equiv (\mathbf{Y}\mathbf{Y}^{\mathrm{T}})/N$;
2) $\widehat{\mathbf{R}}_n := 1/N \sum_i (\widehat{\boldsymbol{\xi}}_i \widehat{\boldsymbol{\xi}}_i^{\mathrm{T}})$; $\{\widehat{\boldsymbol{\xi}}_i$ is given by (10)$\}$
3) $\widehat{\mathbf{R}}_x := 1/N \sum_i ((\mathbf{y}_i - \widehat{\boldsymbol{\xi}}_i)(\mathbf{y}_i - \widehat{\boldsymbol{\xi}}_i^{\mathrm{T}}))$; {estimate of $\widehat{\mathbf{R}}_x$}
4) $\mathbf{E} := [\mathbf{e}_1, \ldots, \mathbf{e}_L]$; $\{\mathbf{e}_i$ are the eigenvectors of $\widehat{\mathbf{R}}_x\}$
5) $\boldsymbol{\delta} := [\delta_1, \ldots, \delta_L]$; $\{\delta_i$ is given by (22)$\}$
6) $(\widehat{\boldsymbol{\delta}}, \widehat{\pi}) := \mathrm{sort}(\boldsymbol{\delta})$;
   {*sort* δ_i by ascending order; *save* the permutation $\widehat{\pi}\}$
7) $\widehat{k} :=$ number of terms $\widehat{\delta}_i < 0$;
8) $\widehat{X} = \langle [\mathbf{e}_{\widehat{i}_1}, \ldots, \mathbf{e}_{\widehat{i}_k}] \rangle$; {signal subspace}

Fig. 3.2. Algoritmo de estimación del subespacio [13]

En consecuencia, se puede apreciar que, de forma genérica, las funciones que pueden extraerse del algoritmo HYSIME son las siguientes:

- Funciones de manejo de matrices: multiplicación entre matrices, inversa y transpuesta de una matriz, etc.
- Funciones para obtener los autovectores de una determinada matriz.

Para concluir la explicación del algoritmo HYSIME, cabe destacar que la mayor ventaja que este algoritmo presenta radica en que, a diferencia de VD, es completamente automático, puesto que no necesita ningún parámetro adicional a la imagen hiperespectral para realizar su función. Esto es algo muy deseable en el desarrollo de la cadena de procesado de imágenes hiperespectrales que se va a explicar en este Proyecto, puesto que supone una simplificación de la interfaz de comunicaciones entre actores.

Por último, antes de pasar a la siguiente fase de la cadena, se va a proporcionar una comparativa entre estos dos algoritmos descritos, con el objetivo de elegir el que se estudiará en profundidad para empezar a construir la librería y, por tanto, se implementará posteriormente como primera parte de la cadena de procesado. Cabe destacar que esta fase se ha considerado opcional dentro del proyecto, por lo que, de forma genérica, no se atendrá a los mismos criterios que la fase de estimación de abundancias.

Como se acaba de mencionar, una de las diferencias principales entre VD y HYSIME es la interfaz de entrada: mientras que HYSIME sólo necesita la imagen hiperespectral original, VD necesita, además, un parámetro , P_F, que determina la sensibilidad del método. Esto supone un gran problema a la hora de implementar dicho algoritmo, puesto que no existe un valor específico para este parámetro. Lo único que especifican sus autores en [74] es que debe estar comprendido entre 10^{-1} y 10^{-5}, pero no se aporta ninguna información adicional acerca de en qué grado repercute dicho valor ni qué otros parámetros han de tenerse en cuenta al fijar P_F.

Además, otra de las diferencias que pueden inferirse del estudio realizado es la complejidad de análisis e implementación de dichos algoritmos. Del algoritmo HYSIME puede observarse que, independientemente de la complejidad computacional -que será el siguiente punto a analizar-, la complejidad de implementación del algoritmo es relativamente sencilla, puesto que, como ya se

ha mencionado, el algoritmo se reduce, mayoritariamente, a operaciones aritméticas entre matrices. De estas operaciones, las que mayor complejidad asociada llevan son la inversa y la obtención de los autovectores del conjunto. Por su parte, el algoritmo VD se desarrollaba mediante la aplicación del llamado método *detector de Neyman - Pearson*. Del estudio de su estructura, que puede observarse en [74], se infiere que su dificultad de implementación es notoriamente superior al HYSIME; además, también puede deducirse que su aportación a la librería a desarrollar en este Proyecto también es sustancialmente menor, puesto que las funciones que se pueden extraer de él son mucho más específicas que las de HYSIME y, en consecuencia, no serán de gran utilidad en una librería de funciones genéricas para el procesado de imágenes hiperespectrales.

El último parámetro que se va a analizar es el tiempo de ejecución. Para ello, se volverá a utilizar el estudio desarrollado por Sergio Sánchez en su tesis doctoral [11], que ya fue citado en el capítulo anterior de este documento. En esta tesis, el autor realizaba un estudio sobre el tiempo computacional empleado por distintos algoritmos de la cadena de desmezclado espectral, tanto en una plataforma mononúcleo como en una multinúcleo. Los dos algoritmos que aquí se han tratado han sido analizados en dicha tesis, por lo que se reproduce aquí un pequeño ejemplo de los resultados obtenidos para ambos.

La tabla 3.1 recoge un resumen de los resultados obtenidos en [11]. Como puede observarse, para cada algoritmo se recogen cuatro resultados: dos de ellos se corresponden con una implementación en serie y otros dos con una implementación en paralelo. A su vez, cada una de esas dos categorías se divide en otras dos, en función del compilador utilizado (*gcc* o *icc*). De forma genérica, de los resultados recogidos en esta tabla puede inferirse que el algoritmo VD es mucho más eficiente en cuanto a tiempo de computación, puesto que, para cualquiera de los parámetros medidos, sus resultados son siempre mejores que los de HYSIME.

	SERIE		**PARALELO**	
	gcc	***icc***	***gcc***	***icc***
VD	5.555 s	3.940 s	0.423 s	0.441 s
HYSIME	28.278 s	20.106 s	1.661 s	1.662 s

Tabla 3.1. Resultados de VD y HYSIME [11]

En conclusión, de esta comparativa se puede extraer que el algoritmo VD es mucho más eficiente desde el punto de vista de velocidad de computación, mientras que el algoritmo HYSIME presenta mejores características desde el punto de vista de sencillez en la interfaz de entrada y complejidad de implementación.

Además, un aspecto clave que favorece a este último algoritmo es que su implementación beneficia a la construcción de la librería de una forma más directa que VD, puesto que, como se ha mencionado anteriormente, sus funciones son más genéricas.

Por último, otro aspecto que ha de tenerse en cuenta es que esta fase de la cadena tiene una naturaleza opcional en este Proyecto y, como tal, no se implementará dentro de la cadena de procesado hiperespectral sino que se considerará como una fase previa de configuración; en consecuencia, el argumento de la velocidad de computación, en este caso particular, no se considera un factor crítico.

Por ello, a la vista de los resultados obtenidos en esta comparativa y tras haber consensuado la elección con el tutor del Proyecto, se ha decidido implementar el algoritmo HYSIME.

Estimación de abundancias

Tal y como se estableció en el capítulo anterior, el objetivo de esta fase es la estimación de la proporción en que cada *endmember* está presente en cada píxel de la imagen original. Al igual que los algoritmos explicados en la fase anterior, los que implementan esta fase también presentan una interfaz genérica común: a la entrada, reciben tanto la imagen original - o la imagen reducida, en caso de estar implementada la fase de reducción dimensional - como el conjunto de *endmembers* extraídos en la tercera fase de la cadena de análisis (extracción de *endmembers*); a la salida, generan un mapa de abundancia asociado a cada *endmember*, por lo que se han de obtener tantos mapas de abundancias como *endmembers* se han extraído en la correspondiente fase. Al contrario que en la fase de estimación del número de *endmembers*, en la literatura se han tratado una gran cantidad de algoritmos para la implementación de esta fase. Entre ellos, los algoritmos más utilizados para la implementación de esta fase son UCLS (*UnConstrained Least Squares*), SCLS (*Sum-to-one Constrained Least Squares*), NCLS (*Non - negativity Constrained Least Squares*), FCLS (*Fully Constrained Least Squares*) e ISRA (*Image Space Reconstruction Algorithm*), que van a analizarse a continuación.

Antes de comenzar el estudio de dichos algoritmos, se han de mencionar dos restricciones que, tradicionalmente, se han aplicado a la hora de calcular las abundancias. Estas restricciones son la de la no negatividad y la de la suma unitaria, que se definen en las ecuaciones 3.3 y 3.4, donde α_i denota la fracción de abundancia correspondiente al *endmember i* y *q* representa el número total de *endmembers* presentes en la imagen.

$$Restricción\ de\ no\ negatividad: \qquad \alpha_i \geq 0, \quad i = 1, \dots, q \tag{3.3}$$

$$Restricción\ de\ suma\ unitaria: \qquad \sum_{i=1}^{q} \alpha_i = 1 \tag{3.4}$$

Sin embargo, en la literatura más reciente [11, 17] se han realizado diversas críticas a estas dos restricciones. Por una parte, la condición de suma unitaria se ha considerado irrelevante, puesto que cualquier conjunto adecuado de *endmembers* debería satisfacer automáticamente esta condición. Según los autores de [17], esto se debe a que "α_i, *para i = 1,...,q, representa las fracciones de los* endmembers *presentes en el considerado píxel*" y, por tanto, su suma debería ser obligatoriamente igual a la unidad. Respecto a la restricción de no negatividad, ésta tiene como objetivo evitar la estimación de abundancias negativas, puesto que carecen de sentido físico.

En este sentido, los algoritmos desarrollados para esta etapa pueden implementar las dos restricciones, implementar sólo una de ellas o no implementar ninguna. Por ejemplo, el algoritmo FCLS, tal y como indican sus dos primeras siglas - *fully constrained* o completamente restringido, en español -, impone ambas restricciones. Sin embargo, según Sergio Sánchez [11], " *dichos métodos son computacionalmente costosos y a veces llevan a soluciones erróneas en el caso de que los endmembers no hayan sido identificados de forma correcta*". Debido a esta complejidad computacional, los algoritmos más aplicados actualmente no imponen ninguna restricción, como UCLS - *unconstrained* o no restringido, en español -, o imponen sólo una de ellas, como NCLS, que aplica la de no negatividad, y SCLS, que aplica la de suma unitaria.

Como todos los algoritmos que tratan esta problemática, UCLS - también denominado LSU en la literatura - trata de estimar de forma exacta la proporción en que cada *endmember* se encuentra en la imagen. Dicho de otra forma, podría decirse que pretende reconstruir la imagen

original minimizando el error y utilizando, para ello, un conjunto M de *endmembers* y aplicando el modelo lineal de mezcla, anteriormente explicado. En concreto, el objetivo de este algoritmo es encontrar el valor de abundancia α_i que reconstruye de forma más exacta cada píxel de la imagen, aplicando para ello la distancia de mínimos cuadrados entre el píxel y su versión reconstruida con dicho modelo. Este algoritmo se formaliza en la ecuación 3.5, donde α_{LS} representa la abundancia, M se corresponde con la matriz de *endmembers* e Y_i denota a cada píxel de la imagen [75].

$$\alpha_{LS} = (M^T \cdot M)^{-1} \cdot M^T \cdot Y_i \tag{3.5}$$

La principal ventaja de este algoritmo radica en su simplicidad, puesto que, al no aplicar ninguna restricción, su implementación es rápida y sencilla. Como puede inferirse de la ecuación 3.5, además, las funciones que pueden extraerse de este algoritmo son muy genéricas, ideales para la librería a desarrollar. De hecho, estas funciones ya han sido mencionadas en la fase de estimación de *endmembers*: transpuesta e inversa de una matriz y multiplicación de matrices, principalmente. Por tanto, a falta del análisis de tiempo computacional, este algoritmo parece idóneo para el Proyecto desarrollado en el presente documento.

Una variación de este algoritmo es SCLS, que realiza el mismo proceso, pero imponiendo la restricción de suma unitaria [75]. La solución obtenida puede observarse en las ecuaciones 3.6 y 3.7, donde $U = (1,1,...,1)^T$, con tantos elementos como número de *endmembers*, y donde I es la matriz identidad.

$$\alpha_{SCLS} = P_{M,1} \cdot \alpha_{LS} + (M^T \cdot M)^{-1} \cdot U \cdot [U^T \cdot (M^T \cdot M)^{-1} \cdot U]^{-1} \tag{3.6}$$

$$P_{M,1} = I - (M^T \cdot M)^{-1} \cdot U \cdot [U^T \cdot (M^T \cdot M)^{-1} \cdot U]^{-1} \cdot U^T \tag{3.7}$$

Como puede observarse, este algoritmo es más costoso que UCLS, puesto que su complejidad computacional es mayor. También cabe destacar que, de forma genérica, las funciones que pueden extraerse de él ya se han mencionado en apartados anteriores: suma, resta y multiplicación de matrices, inversa, traspuesta, etc.

Además, uno de los problemas que arrojan los dos algoritmos mencionados es que no dan solución al problema de las abundancias negativas que, como se ha mencionado anteriormente, no tienen sentido desde el punto de vista físico. Una solución a este problema es el algoritmo NCLS que, al igual que SCLS, es una evolución del UCLS, pero imponiendo la restricción de no negatividad. Sin embargo, a pesar de resolver el problema de las abundancias negativas, añade otros; por ejemplo, este algoritmo ya no es directo, sino que es iterativo, lo que supone un problema porque los algoritmos iterativos suelen presentar elevados tiempos de ejecución. Esto puede observarse en las ecuaciones 3.8 y 3.9 [75], que reflejan el resultado de aplicar al algoritmo UCLS la restricción de no negatividad, y donde α_{NCLS} representa la abundancia y λ denota al llamado vector multiplicador de *Lagrange*. Como se puede apreciar, estas ecuaciones son iterativas, puesto que los dos parámetros mencionados aparecen en ambas y, en consecuencia, dependen la una de la otra.

$$\alpha_{NCLS} = (M^T \cdot M)^{-1} \cdot M^T \cdot Y_i - (M^T \cdot M)^{-1} \cdot \lambda = \alpha_{LS} - (M^T \cdot M)^{-1} \cdot \lambda \tag{3.8}$$

$$\lambda = M^T \cdot (Y_i - M \cdot \alpha_{NCLS}) \tag{3.9}$$

Además, este algoritmo ya no sólo requiere la matriz original y los *endmembers* como entrada, sino que necesita, además, otros dos parámetros adicionales: un umbral de error, ε, y un umbral de parada, ρ [11, 76], puesto que son necesarios para el correcto funcionamiento del algoritmo iterativo. En concreto, este algoritmo ha de seguir repitiéndose mientras se supere el umbral de

error - es decir, que el error sea superior a ε - o mientras no se cumpla la condición de parada - esto es, cuando las abundancias calculadas en dos iteraciones consecutivas presentan una desviación menor que la marcada por ρ-. Cabe destacar que la convergencia de este algoritmo ha sido demostrada en [76].

En consecuencia, respecto a este algoritmo se puede concluir que, aunque mejora la bondad del UCLS, sin embargo aumenta considerablemente el tiempo de ejecución, puesto que su complejidad computacional es muy superior a la del UCLS, debido, principalmente, a que su ratio de convergencia es lento. Al final de este estudio se aportará, al igual que en la fase anterior, una comparativa en la que se analizarán cuantitativamente los tiempos de ejecución para cada algoritmo.

Dentro de la familia de algoritmos que imponen la restricción de la no negatividad, además del NCLS cabe destacar el algoritmo ISRA. Este algoritmo es muy similar al NCLS, pero su paralelización es más sencilla, lo que supone una ventaja con respecto a la consecución del tiempo real. Al igual que NCLS, ISRA también necesita un parámetro adicional para su funcionamiento, que es el umbral de error, ε, ya mencionado anteriormente. La descripción de este algoritmo puede encontrarse tanto en [14] como en [77].

Otra solución que ha surgido con la evolución de estos tres algoritmos explicados es FCLS, que aúna ambas restricciones de forma simultánea. En otras palabras, el algoritmo FCLS supone la fusión de los algoritmos SCLS y NCLS. En concreto, la solución más extendida es extender el algoritmo NCLS, añadiéndole la restricción de la suma unitaria. Esta solución se explica detalladamente en [75]. De forma genérica, una de las soluciones que se han desarrollado para aplicar en NCLS la restricción de la suma unitaria es, básicamente, normalizar las abundancias, de tal forma que todos estén comprendidos entre 0 y 1. Como ya se mencionó anteriormente, si los *endmembers* están correctamente extraídos, la suma de todas las abundancias debe ser obligatoriamente la unidad. De esta definición, se puede deducir que este algoritmo llevará asociado un tiempo de ejecución aún mayor que NCLS, puesto que realiza las mismas operaciones que éste y, además, tras ello debe realizar una normalización de los resultados obtenidos.

Para concluir este apartado, se va a realizar una comparativa de los algoritmos estudiados para, al igual que se hizo con la fase de estimación de *endmembers*, elegir el más adecuado para este Proyecto, que se implementará en apartados posteriores de este documento. Dicha comparativa se basará en los mismos criterios que se utilizaron en la fase anterior: grado de complejidad del algoritmo, calidad de la aportación a la librería y tiempo de ejecución. En este caso, el criterio del tiempo computacional será el más importante a tener en cuenta, puesto que esta fase sí es obligatoria y forma parte de la cadena de desmezclado espectral y, en consecuencia, persigue el objetivo de tiempo real marcado en el capítulo 1 del presente trabajo.

Respecto a la complejidad computacional, ésta se ha mencionado en la explicación de cada algoritmo: se ha visto que UCLS implicaba un grado de complejidad muy bajo, puesto que el algoritmo se realizaba con muy pocas operaciones matriciales básicas; también se ha visto que SCLS y NCLS se construyen añadiendo restricciones a dicho algoritmo, por lo que su grado de complejidad será mayor. De entre ellos, el que presenta un incremento sustancial de la complejidad es NCLS, ya que se trata de un proceso iterativo. Por último, como se acaba de mencionar, FCLS es el algoritmo que mayor grado de complejidad presenta. Por tanto, respecto a este criterio, el algoritmo óptimo es UCLS (o también conocido como LSU).

Por otra parte, respecto a la calidad de la aportación que cada algoritmo hace a la librería, se ha de mencionar que, en este caso particular, el grueso de los algoritmos realiza una aportación muy similar, puesto que, como se ha visto en el estudio de los mismos, éstos están intrínsecamente relacionados y, en consecuencia, las operaciones que realizan son muy similares. Además, cabe destacar que dichas operaciones son muy genéricas, por lo que su aportación a la librería es muy buena. En consecuencia, no se puede establecer un único algoritmo óptimo desde este punto de vista.

El último factor que se va a analizar es el tiempo de ejecución, proporcionando resultados cuantitativos extraídos de diversos estudios consultados. Uno de estos estudios es, de nuevo, la tesis doctoral de Sergio Sánchez [11], que desarrolla el mismo análisis explicado en la fase anterior, pero con los algoritmos UCLS y NCLS. Algunos de los resultados que el autor ha obtenido se recogen en la tabla 3.2: como se puede observar, en comparación, el algoritmo NCLS presenta un tiempo de ejecución sustancialmente mayor que UCLS, especialmente cuando la ejecución se realiza en un sistema mononúcleo. Respecto al algoritmo FCLS, a pesar de que no se aportan tiempos, éstos pueden considerarse similares al NCLS, incluso superiores, por las razones mencionadas anteriormente. Por otra parte, de forma análoga al FCLS, del algoritmo SCLS no se proporcionan tiempos computacionales, pero también puede estimarse su valor entre los tiempos de UCLS y de NCLS, ya que su complejidad así lo indica.

	SERIE		PARALELO	
	gcc	*icc*	*gcc*	*icc*
UCLS	0.313 s	0.251 s	0.045 s	0.041 s
NCLS	92.607 s	74.041 s	0.987 s	0.987 s

Tabla 3.2. Comparativa tiempos UCLS y NCLS [11]

Al igual que en [11], en [14] se proporciona una comparativa, pero en esta ocasión los algoritmos que se comparan son UCLS e ISRA. Ambos algoritmos se han ejecutado en una plataforma mononúcleo, a través de la herramienta software que el autor desarrolla en dicho documento. Los resultados obtenidos se recogen en la tabla 3.3; como se puede observar, al igual que en el análisis anterior, el algoritmo más eficiente en cuanto a tiempo computacional es LSU, puesto que ISRA presenta unos tiempos muy superiores al mismo - con una diferencia de más de dos órdenes de magnitud-.

Algoritmos	Imagen sintética	AVIRIS Cuprite
UCLS - LSU	0.107 s	2.212 s
ISRA	20.84 s	457.2 s

Tabla 3.3. Comparativa de algoritmos UCLS e ISRA [14]

La tabla 3.4 recoge un resumen de la comparativa realizada en este apartado. Como puede observarse, el algoritmo que mejor prestaciones aporta respecto a las categorías analizadas es el UCLS. En consecuencia, éste será el algoritmo que se utilizará para ampliar la librería y, posteriormente, se implementará como última fase de la cadena.

Algoritmos	Complejidad	Aportación a la librería	Tiempo de ejecución
UCLS - LSU	Muy baja	Muy buena	Muy bajo
SCLS	Baja	Muy buena	-
NCLS	Alta	Muy buena	Muy alto
ISRA	Alta	Muy buena	Muy alto
FCLS	Muy alta	Muy buena	Muy alto

Tabla 3.4. Comparativa de algoritmos

3.3. Procedimiento de desarrollo

Tras haber elegido y estudiado los algoritmos a implementar en la cadena de procesado, en este apartado se va a describir, en primer lugar, el proceso de construcción de la librería, detallando las funciones que se han añadido; posteriormente, se pondrán en común dichas funciones con las desarrolladas por Daniel Madroñal en su Proyecto Fin de Grado [1], con el fin de analizar las similitudes y determinar, en consecuencia, la viabilidad de dicha librería; tras ello, para comprobar el funcionamiento de la misma, se procederá a implementar la cadena de procesado en RVC- CAL, haciendo especial hincapié en la construcción de las interfaces de entrada y salida y de la comunicación entre actores; por último, se estudiarán, en función de los actores implementados, las distintas distribuciones que pueden aplicarse para la división entre núcleos, de cara a buscar la combinación que proporcione un menor tiempo de ejecución.

Junto a la lectura de este documento, se recomienda consultar también el proyecto de Daniel Madroñal, puesto que en él se desarrollan varios aspectos relevantes y complementarios de este Proyecto y que, además, serán tenidos en cuenta a lo largo de este capítulo y, sobre todo, en el capítulo 4 - análisis de los resultados obtenidos-. En concreto, los aspectos que se estudian en [1] son, por un lado, el proceso de importación de librerías externas en RVC-CAL, de forma genérica, y, de forma específica, la solución desarrollada para esta librería, que combina funciones en C y C++ en la misma librería; y, por otro lado, el proceso de incorporación de la herramienta PAPI a la cadena de desmezclado espectral, sirviéndose, para ello, de la aplicación desarrollada por Alejo Iván Arias, compañero del GDEM.

Funciones desarrolladas

A continuación, se van a detallar las funciones extraídas para cada uno de los dos algoritmos que han sido elegidos en el apartado anterior, y que se han estudiado en profundidad para entender su funcionamiento y, en consecuencia, poder construir correctamente la librería. Como ya se avanzó en el apartado anterior, el grueso de estas funciones son operaciones genéricas de manejo de matrices, así como operaciones matemáticas con las mismas.

En primer lugar, se describirán las funciones desarrolladas para la fase de estimación de *endmembers*, tras haber estudiado en profundidad el algoritmo HYSIME. Dichas funciones aparecen recogidas en la tabla 3.5, donde se proporciona también una breve descripción. Como se puede observar en dicha tabla, tal y como se esperaba, las funciones extraídas tienen que ver, en su mayoría, con operaciones con matrices, así como con el manejo de las mismas. En concreto, estas funciones pueden clasificarse en tres categorías:

- Operaciones con matrices y vectores. En esta categoría se inscriben las siguientes funciones: *transpose, matrix_mult, matrix_scale, vector_minus_vector, get_inverse, get_eigenvectMatrix, matrix_mult_vector, matrix_plus_value, matrix_minus_matrix* y *vector_mult_matrix*.
- Manejo de matrices. Dentro de esta categoría, se encuentran funciones como *matrix_get_row, matrix_set_row* y *get_diagonal*
- Operaciones matemáticas sencillas. Dentro de esta categoría se inscribe la función *elev*.

Otra característica que puede extraerse de estas funciones es que, además de ser muy genéricas, son también relativamente sencillas de implementar. De hecho, las funciones que más dificultad han supuesto son *get_eigenvectMatrix* y *get_inverse* que, como se explica en la tabla 3.5, se encargan de obtener los autovectores y la matriz inversa, respectivamente. Estas funciones son más complejas porque, para desarrollarlas por software, existen numerosas aproximaciones.

Función	Descripción
transpose	Devuelve la transpuesta de la matriz que se pasa como parámetro
matrix_mult	Multiplica dos matrices que se pasan como parámetros
get_diagonal	Obtiene la diagonal de la matriz cuadrada que se pasa como parámetro
matrix_scale	Divide todos los elementos de la matriz que se pasa como parámetro entre un mismo valor, que también debe proporcionarse
elev	Dados una base y un exponente, realiza la operación de potenciación y devuelve el resultado
vector_minus_vector	Dados dos vectores, los resta y devuelve el resultado por referencia a otro vector
get_inverse	Dada una matriz cuadrada, esta función calcula y devuelve su matriz inversa
get_eigenvectMatrix	Dada una matriz cuadrada, esta función calcula y devuelve los autovectores asociados a ella
matrix_mult_vector	Dados una matriz y un vector, esta función multiplica la matriz por el vector y devuelve el resultado en un vector columna
matrix_get_row	Dados una matriz y un índice, esta función devuelve la fila señalada por dicho índice
matrix_set_row	Dados una matriz y un vector, esta función sustituye una fila de la matriz por el vector proporcionado
matrix_plus_value	Dados una matriz y un valor numérico, esta función suma dicho valor a cada elemento de la matriz que se pasa como parámetro
matrix_minus_matrix	Dadas dos matrices, esta función las suma y devuelve el resultado en otra matriz
vector_mult_matrix	Dados un vector y una matriz, esta función multiplica el vector por la matriz, y devuelve el resultado en un vector fila

Tabla 3.5. Funciones de HYSIME

Tras haber consensuado esta problemática con el tutor del Proyecto y con el compañero Daniel Madroñal, se ha decidido emplear la aproximación utilizada por otros estudios, como [14]. Esta solución consiste en utilizar una serie de librerías específicas para procesado de imágenes, como son ITK - *Insight Toolkit* -, OTB - *Orfeo ToolBox* - y VNL -*Vision Numeric Library* -. Uno de los motivos que más ha primado en esta decisión es que, al aplicar la misma solución que se utiliza en [14], los resultados obtenidos en este Proyecto podrán ser comparados directamente con los extraídos de la herramienta de software libre desarrollada en el mencionado estudio. Esta herramienta ha recibido el nombre de *Hypermix,* y proporciona los recursos necesarios para el análisis de una imagen hiperespectral completa. En consecuencia, como se verá en el siguiente capítulo, para comprobar la bondad de la cadena de desmezclado desarrollada en RVC - CAL se analizará la misma imagen en RVC - CAL y en *Hypermix* y se compararán los resultados obtenidos; si estos resultados coinciden, se dará por contrastada la bondad de los algoritmos.

En concreto, esta aproximación utiliza, como se acaba de mencionar, algunos de los recursos proporcionados por librerías específicas de procesado de imágenes, que ofrecen herramientas para el análisis de imágenes médicas, de teledetección o de alta resolución espacial y espectral. Dichas librerías proporcionan funciones específicas para el cálculo de autovalores y autovectores, así como para el desarrollo de operaciones matriciales que impliquen una elevada

complejidad, como la inversa. Sin embargo, la aplicación de esta aproximación genera otra problemática, puesto que estas librerías están codificadas en C++ y, como ya se ha mencionado anteriormente, la librería desarrollada en este proyecto está codificada en C, por lo que resulta necesario encontrar la forma de combinar los dos lenguajes en una misma librería, y que ésta pueda utilizarse en una aplicación codificada en C. La solución alcanzada en el marco de este proyecto para utilizar ambos lenguajes simultáneamente ha sido explicada detalladamente por el compañero Daniel Madroñal en [1], por lo que en el presente documento no se aportará información al respecto. Cabe destacar que el motivo de mantener C como lenguaje de la librería - y no migrar a C++, a la vista de la necesidad de utilizar este lenguaje para varias funciones - se debe, por un lado, a la futura aplicación de la herramienta PAPI para monitorizar el rendimiento, que también está codificada en C; y, por otro, que este lenguaje es el que más se ha utilizado para la paralelización de aplicaciones multimedia en RVC - CAL.

Una vez extraídas y explicadas las funciones del algoritmo HYSIME, se va a proceder a realizar el mismo procedimiento con las del algoritmo UCLS. Al igual que con el algoritmo anterior, en la tabla 3.6 se recoge la relación de las funciones necesarias para implementar dicho algoritmo, así como una descripción de cada una de ellas. Como se veía en la ecuación 3.5, que se repite aquí para mayor comodidad -ecuación 3.10-, la implementación de este algoritmo es muy sencilla, ya que sólo implica tres operaciones: una transpuesta, una inversa y varias multiplicaciones entre matrices.

$$\alpha_{LS} = (M^T \cdot M)^{-1} \cdot M^T \cdot Y_i \tag{3.10}$$

En consecuencia, como se puede observar en la tabla 3.6, este algoritmo necesita únicamente tres funciones para implementarse - las del principio de la tabla-, que se corresponden con las operaciones mencionadas. Respecto a las otras cuatro funciones que aparecen, si bien no se utilizan en este algoritmo, sí son utilizadas en el SCLS y el NCLS, por lo que se ha decidido realizar también su implementación. De esta forma, con la librería puede construirse cualquiera de los tres algoritmos mencionados, y no sólo UCLS.

Función	Descripción
transpose	Devuelve la transpuesta de la matriz que se pasa como parámetro
matrix_mult	Multiplica dos matrices que se pasan como parámetros
get_inverse	Dada una matriz cuadrada, esta función calcula y devuelve su matriz inversa
matrix_minus_matrix	Dadas dos matrices, esta función las suma y devuelve el resultado en otra matriz
matrix_plus_matrix	Dadas dos matrices, esta función las resta y devuelve el resultado en otra matriz
matrix_minus_value	Dados una matriz y un valor numérico, esta función resta dicho valor a cada elemento de la matriz que se pasa como parámetro
matrix_plus_value	Dados una matriz y un valor numérico, esta función suma dicho valor a cada elemento de la matriz que se pasa como parámetro

Tabla 3.6. Funciones de UCLS, SCLS y NCLS

Por último, para finalizar esta sección se va a realizar una comparativa entre las funciones desarrolladas para cada fase, con el objetivo de verificar si existen funciones que sean necesarias para ambos algoritmos, de cara a demostrar la necesidad de desarrollar una librería de análisis de imágenes hiperespectrales.

La fase anterior parece arrojar resultados alentadores al respecto, pues, como se acaba de mencionar, para implementar otros dos algoritmos - aparte del utilizado para extraer las funciones - sólo se ha necesitado añadir cuatro funciones. Además, si comparamos las tablas anteriores, observamos que, efectivamente, de las siete funciones explicadas en la tabla 3.6, cuatro de ellas ya estaban presentes en la tabla 3.5; por tanto, a falta de la comparativa con las otras dos fases, que se realizará a continuación, se puede afirmar que, como ya se ha explicado anteriormente, el desarrollo de la librería está completamente justificado.

Unificación de la librería

Una vez expuestas las funciones necesarias para implementar las fases de estimación del número de *endmembers* y de estimación de abundancias, en este apartado se va a proceder a realizar la unificación de la librería. Para ello, se analizarán las funciones desarrolladas por Daniel Madroñal en [1] y se realizará una comparativa entre éstas y las explicadas en este documento, de cara a localizar las funciones que se repitan en ambos proyectos y, así, justificar completamente la necesidad de una librería de análisis de imágenes hiperespectrales.

Las funciones que han sido desarrolladas en [1] para las fases de reducción dimensional y extracción de *endmembers* se recogen en la tabla 3.7. Conviene mencionar que los algoritmos que se han estudiado para la extracción de dichas funciones han sido PCA (*Principal Component Analysis*) y VCA (*Vertex Component Analysis*), respectivamente.

Como se indica en [1], esas dos fases han permitido extraer un total de 24 funciones, de las cuales seis de ellas son comunes a ambos algoritmos, tal y como ha ocurrido con las fases desarrolladas en este proyecto. Además, si se analiza la mencionada tabla, puede observarse que la naturaleza de las funciones es mayoritariamente matemática y matricial, al igual que las funciones de las tablas 3.5 y 3.6. De hecho, se puede mantener la clasificación propuesta en el apartado anterior, dividiendo las funciones en las tres categorías mencionadas.

Una vez recogidas las funciones desarrolladas para las otras dos fases de la cadena, se va a realizar la comparativa entre éstas y las explicadas anteriormente en las tablas 3.5 y 3.6. Tal y como se esperaba, se han detectado un gran porcentaje de funciones que cumplen el mismo objetivo. Para proporcionar esta información de una forma más visual, en la tabla 3.7 se han resaltado las funciones comunes en color verde. Como puede observarse, existen un total de nueve funciones que coinciden entre ambos proyectos. Además, si se atiende a la funcionalidad de las mismas, se puede comprobar que éstas se corresponden con las funciones más básicas del conjunto, lo que refuerza, una vez más, la idea de la necesidad de una librería, ya que comprueba la existencia de funcionalidades comunes entre los algoritmos de procesado de imágenes hiperespectrales más utilizados en la literatura.

Para finalizar esta comparativa, en la tabla 3.8 se proporciona la lista completa de funciones desarrolladas entre ambos proyectos, completando así la unificación de la librería. Para terminar de justificar la necesidad de la librería, en ella se han marcado en color verde aquellas funciones que se han utilizado en más de un algoritmo, independientemente de si éste ha sido analizado en este documento o en [1]. Así, las funciones marcadas en color verde denotan funcionalidades comunes, mientras que las de fondo blanco se corresponden con funciones más específicas, utilizadas en un único algoritmo. Como se puede comprobar, entre ambos proyectos se han desarrollado un total de 32 funciones, de las que 13 son utilizadas en varias fases; esto supone que un 40.6% de las mismas se utilizan en más de un algoritmo. Además, también se pueden observar dos funciones - *matrix_mult* y *transpose*- que, de hecho, se utilizan en todas las fases de la cadena, además de *get_eigenvectMatrix*, que se usa en tres de ellas.

Función	Descripción
transpose	Devuelve la transpuesta de la matriz que se pasa como parámetro
matrix_mult	Multiplica dos matrices que se pasan como parámetros
matrix_scale	Divide todos los elementos de la matriz que se pasa como parámetro entre un mismo valor, que también debe proporcionarse
matrix_extend	Esta función amplía las dimensiones de una matriz, rellenando con ceros las posiciones vacías
matrix_fill_column	Esta función rellena una columna de la matriz que se pasa como parámetro con un mismo valor, que también se proporciona
elev	Dados una base y un exponente, realiza la operación de potenciación y devuelve el resultado
log	Esta función calcula el logaritmo del número que se pasa como parámetro
sq_root	Esta función calcula la raíz cuadrada del número que se pasa como parámetro
rem	Esta función devuelve el resto de la división de dos números, que se proporcionan como parámetros
maxSqrtSum	Esta función eleva al cuadrado los elementos de cada fila, los suma, hace la raíz cuadrada de la suma de cada fila y devuelve la mayor de ellas
mean_vector	Dado un vector, esta función calcula la media del mismo
vector_minus_vector	Dados dos vectores, los resta y devuelve el resultado por referencia a otro vector
vector_minus_value	Dados un vector y un valor, esta función resta dicho valor a cada elemento del vector y almacena el resultado en otro
vector_scale	Esta función divide todos los elementos de un vector entre un mismo valor
get_pinverse	Dada una matriz cuadrada, esta función calcula y devuelve su matriz pseudo - inversa
get_eigenvectMatrix	Dada una matriz cuadrada, esta función calcula y devuelve los autovectores asociados a ella
matrix_mult_vector	Dados una matriz y un vector, esta función multiplica la matriz por el vector y devuelve el resultado en un vector columna
matrix_get_column	Dados una matriz y un índice, esta función devuelve la columna señalada por dicho índice
matrix_set_column	Dados una matriz y un vector, esta función sustituye una columna de la matriz por el vector proporcionado
matrix_get_row	Dados una matriz y un índice, esta función devuelve la fila señalada por dicho índice
matrix_set_row	Dados una matriz y un vector, esta función sustituye una fila de la matriz por el vector proporcionado
matrix_reduce	Esta función trunca las dimensiones de la matriz que se pasa como parámetro
random_vector	Esta función genera un vector de valores aleatorios entre 0 y 1
getIndexMax	Esta función devuelve la posición del elemento con mayor valor absoluto de la matriz que se pasa como parámetro

Tabla 3.7. Funciones de PCA y VCA [1]

Por último, cabe destacar que, si bien las funciones listadas en la tabla 3.8 se corresponden con el total de funciones desarrolladas para los cuatro algoritmos, se ha de mencionar que la librería aún no está completa. Esto se debe a que, tras analizar dicha tabla, se ha decidido añadir otras funciones que, aunque no han sido requeridas por estos algoritmos, son complementarias o análogas a algunas de las incluidas. Éste es el caso, por ejemplo, de la función *vector_mult* que es análoga a *matrix_mult* y su funcionalidad es genérica.

En consecuencia, se ha decidido añadir también este tipo de funciones para completar la librería final, de cara a proporcionar una solución lo más completa posible en lo que se refiere a funciones genéricas de manejo de matrices, facilitando así la implementación de cualquier algoritmo de análisis de imágenes hiperespectrales, y no únicamente los estudiados e implementados en este documento, así como en [1].

Función	Función
transpose	***get_eigenvectMatrix***
matrix_mult	***matrix_mult_vector***
matrix_scale	***matrix_get_column***
matrix_extend	***matrix_set_column***
matrix_fill_column	***matrix_get_row***
elev	***matrix_set_row***
log	***matrix_reduce***
sq_root	***random_vector***
rem	***getIndexMax***
maxSqrtSum	***get_inverse***
mean_vector	***matrix_plus_value***
vector_minus_vector	***matrix_minus_value***
vector_minus_value	***matrix_minus_matrix***
vector_scale	***matrix_plus_matrix***
get_pinverse	***vector_mult_matrix***
get_diagonal	***vector_mult***

Tabla 3.8. Unificación de la librería

Para concluir este apartado, conviene mencionar que, aunque la inclusión de estas funciones completa la librería desde el punto de vista algorítmico, a lo largo de este mismo capítulo se añadirán otras funciones relacionadas con la interfaz de comunicación. La decisión de incluir estas funciones en la librería se debe a que, como el objetivo final es proporcionar una herramienta que facilite el procesado de imágenes hiperespectrales desde RVC - CAL, se ha considerado necesario desarrollar funciones que simplifiquen tanto las interfaces de entrada y salida como la comunicación entre actores, de cara a minimizar la carga de trabajo y el tiempo requerido para la implementación de una cadena de desmezclado en dicho lenguaje.

A lo largo del siguiente apartado se irán explicando dichas funciones, siguiendo una metodología similar a la de este apartado. Asimismo, se volverá a proporcionar una tabla que aúne todas las funciones de la librería, incluyendo éstas últimas, y que supondrá la versión definitiva para la librería desarrollada en este Proyecto.

Comunicación entre actores e interfaz

Una vez completada la librería en su primera versión, se va a proceder a explicar la construcción de la cadena de procesado de imágenes hiperespectrales en RVC - CAL. Para ello, se recomienda primero la consulta del trabajo recogido en [1], puesto que en él se exponen conceptos básicos para la comprensión de este apartado. Por ejemplo, en dicho documento se expone la integración en RVC - CAL de la librería desarrollada, así como la combinación de las funciones implementadas en C y C++ en una misma librería. Utilizando esa información y la recogida en este documento acerca de la unificación de la librería, se procederá a desarrollar la cadena de desmezclado, haciendo especial hincapié en el desarrollo de las interfaces necesarias para la misma.

Para ello, en primer lugar se explicará la estructura elegida para la implementación de dicha cadena, proporcionando los argumentos necesarios para justificar la división en actores realizada; posteriormente, se estudiará la división de cada actor en distintas acciones, analizando el porqué de esta división; y, por último, se explicarán tanto las interfaces de entrada y salida como las de comunicación entre actores, detallando las funciones que han tenido que desarrollarse para ello.

División en actores

Como se comentó anteriormente, se ha decidido implementar cada fase de la cadena de procesado de imágenes hiperespectrales en un actor independiente. Los motivos que se encuentran detrás de esta decisión son, principalmente:

- Facilitar la modificación y/o sustitución de los algoritmos que implementan cada fase de la cadena, sin que por ello se vean afectadas el resto de etapas.
- Reproducir la secuencia natural de las etapas de la cadena, de tal forma que pueda observarse el posible paralelismo entre ellas.

En consecuencia, la red implementada consta, en total, de tres actores, además de los encargados de la interfaz de entrada y salida. El conjunto completo de actores aparece en la figura 3.3: como puede observarse, se han desarrollado tres actores principales - PCA, VCA y LSU - que implementan las fases de reducción dimensional, extracción de *endmembers* y estimación de abundancias, respectivamente; y los actores *Source* y *Display*, que se corresponden con las mencionadas interfaces de entrada y salida.

Como puede observarse en la mencionada figura, existen una serie de arcos o flechas que unen los actores entre sí. Como se mencionaba en el capítulo 2 del presente documento, estas flechas representan los flujos de datos entre actores, y facilitan la extracción del paralelismo implícito en la cadena de desmezclado espectral. Un ejemplo de esto puede apreciarse en esta figura: si se analiza el actor PCA, se puede notar que su puerto de salida se une, simultáneamente, con los puertos de entrada de VCA y LSU. Esto significa que dicha salida se utiliza en ambos actores y, en consecuencia, puede paralelizarse la ejecución de aquellas funciones que la utilicen, en lugar de ejecutarse secuencialmente dichas etapas.

En consecuencia, realizar este tipo de deducciones en un lenguaje tradicional, como C, sería una tarea muy compleja que, como se ve en este ejemplo, en RVC - CAL se ve reducida al análisis de los flujos de datos entre actores, permitiendo así la simplificación de la búsqueda de una división óptima de los actores en los procesadores, de tal forma que la cadena minimice el tiempo de procesado de una imagen hiperespectral, como se verá a lo largo de este apartado.

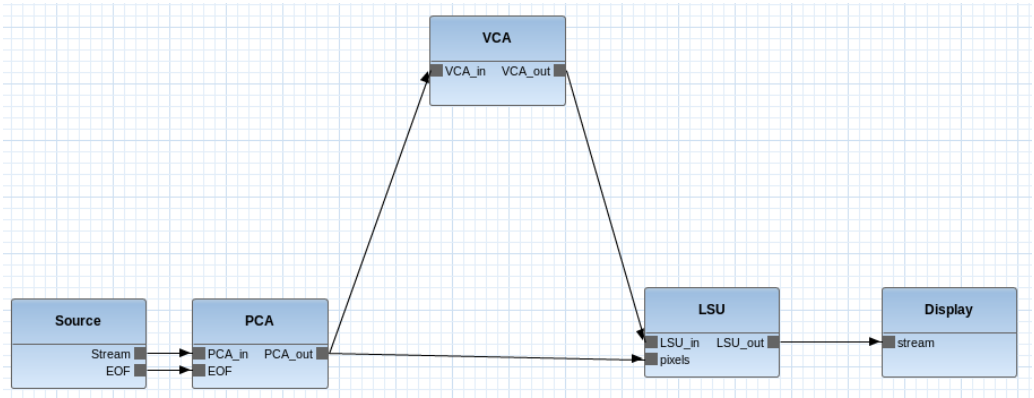


Fig. 3.3. Cadena de desmezclado espectral implementada en RVC - CAL

División en acciones

Una vez justificada la división de actores elegida, se va a resumir brevemente la organización de cada actor de la cadena de desmezclado espectral - PCA, VCA y LSU - en distintas acciones. De forma genérica, estos actores presentan, por un lado, una acción que desarrolla el algoritmo en sí y, por otro, tantas acciones como puertos de entrada y salida presente dicho actor - excepto PCA, que presenta una única acción para sus dos señales de entrada -. En consecuencia, PCA y VCA constan de tres acciones cada uno - una para el algoritmo, otra para el puerto de entrada y otra para el puerto de salida -, mientras que LSU consta de cuatro acciones - una para cada puerto de entrada, otra para el puerto de salida y otra para el algoritmo en sí -. El principal motivo de esta elección de división de acciones se debe a que, de esta forma, se separan de forma concisa las distinta funcionalidades que se presentan en cada uno de los actores, aislando el algoritmo de las tareas relativas a la comunicación. Esto permite, nuevamente, facilitar la modificación y/o sustitución del algoritmo implementado en cada etapa, puesto que, con esta organización, dicha problemática se reduciría a la modificación de una única acción dentro del correspondiente actor.

Para facilitar la comprensión de esta división de acciones, en la figura 3.4 se proporciona el código de las acciones desarrolladas para el algoritmo LSU, que constituye la última etapa de la cadena de desmezclado espectral - estimación de abundancias-. Como se puede observar, dicho algoritmo está compuesto por las cuatro acciones mencionadas:

- *receive_data*: Esta acción se encarga de recibir, del actor *PCA*, la imagen dimensionalmente reducida. Como puede apreciarse en dicha imagen, la funcionalidad de dicha acción es muy simple: va recibiendo los píxeles de la imagen y los va almacenando en una matriz. Esta acción permite observar la utilización de una de las nuevas funciones añadidas a la librería para solucionar problemas de comunicación - *float_to_double* -, que serán explicadas más adelante.
- *receive_endmembers*: Esta acción tiene como objetivo la recepción de los *endmembers* extraídos en el actor *VCA*. Al igual que la acción anterior, ésta va recibiendo datos por el puerto de entrada y los va almacenando en una matriz interna.
- *LSU_algorithn*: Esta acción desarrolla el algoritmo UCLS en sí. Como puede apreciarse, todas las sentencias de la acción son llamadas a la librería desarrollada, lo que ejemplifica la simplificación conseguida con la utilización de la misma.
- *send_abundances*: Por último, esta acción se encarga de enviar al actor *Display* el resultado final de la cadena de desmezclado, que es una matriz con las abundancias calculadas para cada *endmember* extraído en el algoritmo VCA.

De la figura 3.4 pueden extraerse varias conclusiones importantes, como puede ser la facilidad de utilización de la librería, por un lado, y la simplificación de la complejidad de implementación de los algoritmos, por otro. En esta figura también se pone de manifiesto la necesidad de ciertas funciones auxiliares, como *float_to_double* y *write_txt*, que serán explicadas en profundidad a lo largo de este capítulo.

```
receive_data: action pixels: [token] repeat num_PC ==>

guard
    pix_received < (rc)
do
    array_rec_pixels := token;
    foreach int(size= 32)m in 0 .. ((num_PC)-1) do
        image_PIXELS[m][pix_received] := float_to_double(array_rec_pixels[m]);
    end

    pix_received := pix_received + 1;
    if (pix_received = rc) then
        //write_txt(image_PIXELS,num_PC,rc, "PIXELS_MIO_LSU.txt");
    end
end
receive_endmembers: action LSU_in: [token] repeat num_PC ==>

guard
    received < int_Endmembers
do
    array_rec := token;
    foreach int(size= 32)m in 0 .. ((num_PC)-1) do
        image_END[m][received] := float_to_double(array_rec[m]);
    end

    received := received + 1;
    if (received = int_Endmembers) then
        //write_txt(image_END,num_PC,int_Endmembers, "image_END_MIO.txt");
    end
end
```

```
LSU_algorithm: action ==>

guard
    received >= int_Endmembers and pix_received >= rc and lsu_done = false
do
    println("****Inicio LSU****");
    tINI_LSU := get_time();
    transpose(image_END,image_END_Transp, num_PC, num_Endmembers);
    //write_txt(image_END_Transp, int_Endmembers, num_PC, "END_T_MIO.txt");
    matrix_mult(image_END_Transp, image_END, PRODUCT, num_Endmembers, num_PC, num_PC, num_Endmembers);
    //write_txt(PRODUCT, int_Endmembers, int_Endmembers, "PRODUCT_MIO.txt");
    get_inverse(PRODUCT, INVERSE, int_Endmembers, int_Endmembers);
    //write_txt(INVERSE, int_Endmembers, int_Endmembers, "INVERSE_MIO.txt");
    matrix_mult(INVERSE, image_END_Transp, COMPUT_MATRIX, num_Endmembers, num_Endmembers, num_Endmembers,
    //write_txt(COMPUT_MATRIX, int_Endmembers, num_PC, "COMPUT_MATRIX_MIO.txt");
    matrix_mult(COMPUT_MATRIX, image_PIXELS, ABUNDANCES, num_Endmembers, num_PC, num_PC, rc);
    //write_txt(ABUNDANCES, int_Endmembers, rc, "ABUNDANCES_MIO.txt");
    tEND_LSU := get_time();
    tDIFF_LSU := tEND_LSU - tINI_LSU;
    println("El tiempo total de LSU es: " + tDIFF_LSU);
    println("****Fin LSU****");
    println("****Imagen procesada****");
    lsu_done := true;
end

send_abundances: action ==> LSU_out: [output] repeat num_Endmembers

guard
    lsu_done = true and token_sent < rc
do

    foreach int(size= 32)m in 0 .. (num_Endmembers-1) do
        array_env[m] := double_to_float(ABUNDANCES[m][token_sent]);
    end

    output := array_env;
    //println("****Envío VCA_out!****" +"         " + token_sent);
    token_sent := token_sent + 1;
end
```

Fig. 3.4. Acciones del actor LSU

Asimismo, en la figura 3.5 se proporciona una captura del fichero utilizado en RVC - CAL para la importación de la librería. Esta captura muestra la forma en que se integra la librería desarrollada en C en este nuevo lenguaje: se ha de crear un fichero en el que se indiquen los prototipos de las funciones de la librería, precediéndolos con el prefijo *@native function* o *@native procedure*, dependiendo de si la función devuelve algún parámetro o no.

Como puede apreciarse, este fichero se incluye dentro de un paquete (sentencia *package hsi_analysis*); de esta forma, los actores que quieran utilizar este fichero sólo tienen que importar dicho paquete (sentencia *import hsi_analysis.hsi_analysis.**) para poder utilizar las funciones desarrolladas en este proyecto, de forma análoga a cualquier otra librería, tal y como aparece en el código del actor LSU mostrado anteriormente.

```
package hsi_analysis;
unit hsi_analysis:
    uint(size=32) MAX = 10000;
    uint(size=32) rowsMax = 4096;
    uint(size=32) columnsMax = 4096;
    uint(size=32) bandsMax = 4096;

    /*File reading and writing */

    @native procedure gen_conf_data(int(size=32) rows, int(size=32) columns, int(size=32) bands, int(
    end

    @native procedure write_short_binary(int(size=16) matrixOut[MAX][MAX], int(size = 32) rc, int(siz
    end

    @native procedure write_double_binary(double matrixOut[MAX][MAX], int(size = 32) rc, int(size = 3
    end

    @native procedure gen_hdr(double matrixOut[MAX][MAX], int(size=32) rows, int(size=32) columns, in
    end

    @native procedure read_txt(double matrixIn[MAX][MAX], int(size = 32) rows, int(size = 32) columns
    end

    @native procedure write_txt(double matrixOut[MAX][MAX], int(size = 32) rows, int(size = 32) colum
    end

    /*Algorithmics */

    @native function mean_vector(double vector[MAX], int(size = 32) positions) --> double
    end

    @native procedure vector_minus_value(double vectorIn[MAX], double value, double vectorMinus[MAX],
    end
```

Fig. 3.5. Importación de la librería en RVC - CAL

Interfaces y comunicación

Una vez explicada y justificada tanto la división de actores como la de acciones, se va a proceder a completar el estudio de la cadena de análisis de imágenes hiperespectrales, analizando, para ello, el desarrollo de las interfaces de entrada y salida, así como la resolución de los problemas de comunicación surgidos durante el desarrollo de este trabajo.

Como se mencionó anteriormente, el lenguaje RVC - CAL se compone de una serie de nodos o actores que se relacionan entre sí, intercambiándose paquetes de datos. En consecuencia, en las redes o *networks* implementadas en este nuevo lenguaje existe una comunicación que debe ser diseñada de forma eficiente.

Como podía observarse en la *network* desarrollada en este trabajo, se necesita una interfaz de entrada, una interfaz de salida y una comunicación entre los cinco actores que conforman el sistema. A continuación, se explica la funcionalidad de cada una de ellas, así como su implementación y si se ha necesitado, para ello, añadir alguna función a la librería.

Interfaz de entrada

El objetivo de esta interfaz es leer una imagen hiperespectral y enviársela al actor *PCA*, para que éste la almacene en una matriz. Para desarrollar esta interfaz, se han estudiado los formatos de almacenamiento de estas imágenes. De forma genérica, se pueden definir tres formatos principales, denominados BIL -*Banda Intercalada por Línea*-, BIP -*Banda Intercalada por Píxel*- y BSQ -*Banda Secuencial*-:

– *BIL*: Este formato almacena la información del píxel banda por banda para cada línea de la imagen. En la figura 3.6 se proporciona un diagrama ilustrativo del formato BIL para una imagen RGB (tres bandas).

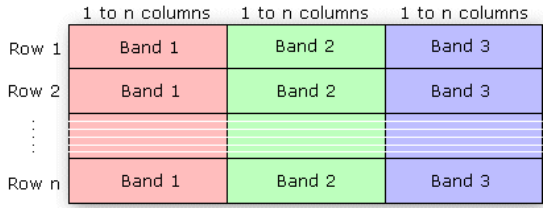


Fig. 3.6. Formato BIL[16]

– *BIP*: Este formato almacena, para cada píxel, la información de todas las bandas en una misma columna. En la figura 3.7 se proporciona un diagrama ilustrativo del formato BIP para una imagen RGB.

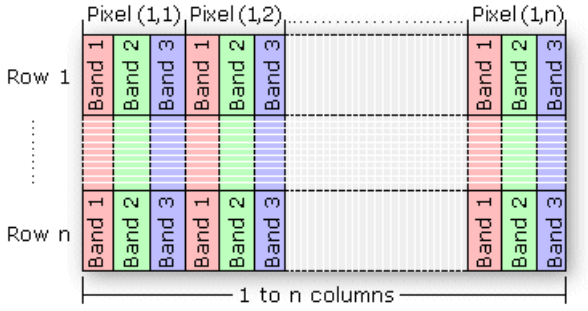


Fig. 3.7. Formato BIP[16]

– *BSQ*: Este formato almacena los datos de forma opuesta al formato BIP: si éste almacenaba la información de todas las bandas para un mismo píxel, el formato BSQ almacena la información relativa a cada banda de todos los píxeles. De esta forma, la información de todos los píxeles para la banda 1 se almacena primero, después se almacenan los de la banda 2, y así sucesivamente. En la figura 3.8 se proporciona un ejemplo del formato BSQ para una imagen RGB.

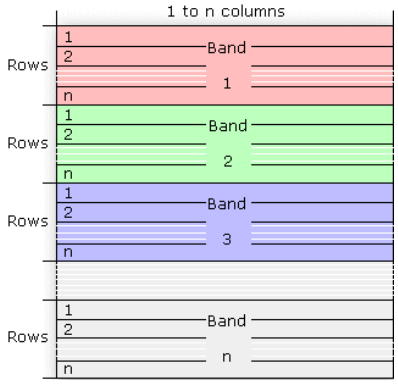


Fig. 3.8. Formato BSQ[16]

[16] http://help.arcgis.com/es/arcgisdesktop/10.0/help/index.html#/na/009t00000010000000/

Para concluir la explicación de los tres formatos, en la figura 3.9 se proporciona un ejemplo de imagen RGB, tanto descompuesta en cada una de las tres bandas como la imagen total. Utilizando esta imagen, se va a proporcionar un ejemplo de cómo quedaría el fichero con dicha imagen, aplicando cada una de las distintas estructuras explicadas en la página anterior. Cabe mencionar que, aunque las imágenes hiperespectrales se almacenan en ficheros binarios, la reproducción de la imagen de tres bandas se realizará en caracteres ASCII para facilitar su comprensión.

BIL: 0 0 0 0 0 0 0 0 0 0 64 64 128 128 255 255 255 255 255 255 255 255 255 255...

BIP: 0 0 255 0 0 255 0 64 255 0 64 255 0 128 255 0 128 255 0 255 255 0 255 255...

BSQ: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ...

Como puede observarse, en el formato BIL se escribe primero la primera línea de la primera banda (banda roja), después la primera línea de la segunda banda (banda verde) y, posteriormente, la primera línea de la tercera banda (banda azul), y así sucesivamente. Por su parte, en el formato BIP se escribe, para cada píxel, los tres valores rojo - verde - azul (por ejemplo, primer píxel: 0 0 255), empezando por el píxel de la esquina superior izquierda. Por último, se puede apreciar que, en el caso del formato BSQ, se escribe, línea a línea, la información de toda la banda roja, después la de toda la banda verde y, por último, la de toda la banda azul.

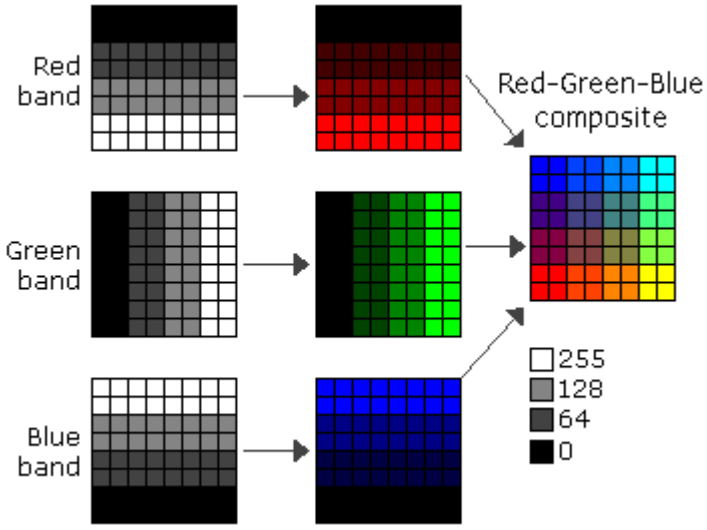


Fig. 3.9. Imagen RGB[17]

Una vez estudiados los distintos formatos de almacenamiento de imágenes hiperespectrales, se ha decidido implementar un método que lea un fichero organizado con la estructura BSQ. El motivo de esta elección se basa en que las imágenes hiperespectrales que se utilizarán en el siguiente capítulo están ordenadas siguiendo esta estructura. En esta primera versión sólo se ha implementado la lectura en este formato, si bien se añadirán los otros dos formatos en futuras versiones. Esta lectura ha sido implementada en el actor *Source* de la cadena; conviene destacar que en este desarrollo ha colaborado el compañero Ángel Villanueva.

Esta interfaz de entrada lee la imagen hiperespectral en formato BSQ y se la envía al actor *PCA*. Sin embargo, se ha de resaltar que este envío se realiza en formato BIP - es decir, el actor *PCA* recibe los datos de todas las bandas de cada píxel-, por los motivos que se explican a continuación.

[17] http://help.arcgis.com/es/arcgisdesktop/10.0/help/index.html#/na/009t00000010000000/

Comunicación *Source - PCA*

Como se acaba de explicar, el actor *Source* lee la imagen hiperespectral del fichero con formato BSQ, pero se la envía al actor *PCA* con formato BIP. La principal razón que ha motivado este cambio en la estructura a la hora de enviar los datos al siguiente actor de la cadena se basa en la facilitación de la recepción de estos datos en dicho actor; el algoritmo PCA necesita tener la imagen hiperespectral almacenada en una matriz cuyas dimensiones sean *píxeles* x *bandas*, por lo que, si se envían los datos de todas las bandas para un mismo píxel, la matriz puede irse rellenando fila a fila, de forma secuencial, simplificando así la acción de recepción del actor *PCA*. Sin embargo, esto conlleva un ligero incremento de la dificultad de implementación de la interfaz de entrada, puesto que se ha de realizar el cambio de estructura al leer.

No obstante, el principal problema que se presenta en esta comunicación es el gran volumen de datos a transmitir. Por ejemplo, una de las imágenes que se va a utilizar para comprobar el correcto funcionamiento de la cadena es una imagen de 10000 píxeles y 221 bandas, lo que arroja un total de 2210000 elementos a enviar de un actor a otro. Parece evidente que, si uno de los objetivos es tratar de alcanzar el tiempo real, se ha de intentar minimizar el tiempo de comunicación al mínimo posible. Por ello, enviar estos elementos uno a uno no parece la solución óptima, puesto que tener que ejecutar dos acciones (la de envío y la de recepción) más de dos millones de veces introducirá un tiempo de retardo intolerable. La solución que se ha dado a este problema es crear un *buffer* en ambas acciones y, en lugar de enviar elemento a elemento, esperar a que este buffer se llene y enviarlo como un único elemento o *token*. Ahora, el siguiente problema es decidir qué tamaño ha de tener este *buffer*, puesto que si es muy pequeño no soluciona el problema y, si es muy grande, puede desbordar y bloquear los procesadores. En este trabajo, se ha decidido establecer el tamaño de este *buffer* al número de bandas de la imagen; de esta forma, el *buffer* enviado se corresponderá con cada fila de la matriz de recepción, y habrá que enviar un total de 10000 elementos.

Por último, conviene mencionar que, en la implementación de esta interfaz de comunicación, se ha necesitado añadir dos funciones auxiliares a la librería. Estas funciones proporcionan las herramientas necesarias para convertir un dato de tipo *double* a *float*, y viceversa; y han sido desarrolladas porque RVC - CAL no permite que los actores se intercambien datos de tipo *double*, pero sí de tipo *float*. Como tanto en la interfaz de entrada como en el PCA la imagen hiperespectral ha de ser tipo *double* (para poder almacenar números decimales), antes de enviar un *token* y, tras recibirlo, se ha de cambiar el tipo de dato, utilizando para ello las funciones desarrolladas (*float_to_double* y *double_to_float*). Cabe destacar que la conversión entre tipos de datos puede añadir un pequeño error de redondeo en los datos intercambiados. De forma genérica, este error podrá considerarse despreciable; no obstante, en el siguiente capítulo se estudiará en profundidad su impacto en el resultado final.

Comunicación *PCA - VCA* y *PCA - LSU*

Las interfaces de comunicación entre los actores *PCA* y *VCA* y los actores *PCA* y *LSU* son idénticas entre sí y, a su vez, muy similares a la que se acaba de explicar. La única diferencia entre ellas es el volumen de datos a enviar y, en consecuencia, el número de veces que se repiten las acciones de envío y recepción. De forma genérica, el actor *PCA* proporciona a su salida la imagen reducida dimensionalmente, que está almacenada en una matriz de dimensiones *píxeles* x *bandas reducidas*. En consecuencia, el volumen de datos a enviar será el resultado de ese producto; si, por ejemplo, se desea reducir la imagen a diez bandas, el número total de datos a intercambiar será de 100000 elementos. Además, en esta interfaz también se ha de optimizar el

tiempo de comunicación, por lo que se aplica la misma técnica explicada en el apartado anterior: la acción de envío almacena los datos en un *buffer* y, cuando éste se llena, se envía como un solo elemento o *token*. En este caso, el valor elegido para establecer el tamaño del *buffer* ha sido el número de bandas reducidas. La razón para elegir este valor es la misma que en el caso anterior: dicho valor constituye una de las dimensiones de la matriz en la que se reciben los datos transmitidos en los actores receptores, lo que facilita su almacenamiento en la misma.

Cabe destacar que, aunque la salida del actor *PCA* es utilizada por dos actores distintos, la acción de envío en dicho actor es única. Éste es uno de los puntos de la cadena donde se puede observar el paralelismo inherente, puesto que, si se implementase de forma secuencial, la imagen reducida tendría que enviarse al actor *VCA* y éste, a su vez, mandársela al *LSU*. Como puede comprobarse, este camino es mucho más lento, puesto que implica la duplicación de un intercambio de un gran volumen de datos. Por último, conviene resaltar que esta comunicación no requiere el desarrollo de ninguna función adicional, puesto que las únicas que utiliza son las ya mencionadas *double_to_float* y *float_to_double*.

Comunicación *VCA - LSU*

La comunicación que se produce entre el actor *VCA* y el *LSU* tiene como objetivo el envío del conjunto de *endmembers* extraídos en el primer actor al segundo. De nuevo, esta interfaz es muy similar a la de las comunicaciones anteriormente descritas, variando únicamente el tamaño de los datos a enviar. En este caso, dicho tamaño viene dado por el producto *número de* endmembers *extraídos* x *bandas reducidas* que, en general, será muy inferior a los volúmenes ejemplificados anteriormente. Por ejemplo, manteniendo el mismo número de bandas reducidas que en el caso anterior (10), si se extraen 5 *endmembers* el volumen total de datos a transmitir será de 50. Aún así, se ha decidido mantener la metodología de los *buffers* de entrada y salida, con el objetivo de proporcionar homogeneidad a la cadena. En concreto, en este caso el tamaño de los *buffers* será el número de *endmembers* extraídos en el actor *VCA*, y habrá que realizar tantos envíos como bandas reducidas haya. Respecto a las funciones requeridas, cabe destacar que, al igual que en la fase anterior, en ésta no se ha desarrollado ninguna función adicional a las ya explicadas.

Comunicación *LSU - Display* e interfaz de salida

Para concluir el análisis de la comunicación en la cadena, se va a explicar, por un lado, la comunicación entre los actores *LSU* y *Display* y, por otro, la funcionalidad de este último actor, que constituye la interfaz de salida del sistema.

La comunicación *LSU - Display* se explicará de forma sucinta, puesto que es muy similar a las explicadas anteriormente. En este caso, el actor *LSU* tiene que enviar al *Display* la matriz con las abundancias estimadas. Esta matriz tiene un tamaño de *número de* endmembers x *píxeles*; suponiendo que se mantiene el primer parámetro al valor utilizado anteriormente (5), el volumen total de elementos a transmitir será de 50000. Al igual que en las comunicaciones anteriores, se utilizarán los *buffers* de entrada y salida para disminuir el número de elementos a enviar; en este caso, ajustando el tamaño de dichos *buffers* al número de *endmembers*, el volumen total de elementos a enviar se reducirá a 10000 *tokens*. Asimismo, esta comunicación tampoco incorpora nuevas funciones a la librería desarrollada.

Respecto a la interfaz de salida del sistema, la finalidad de la misma es guardar los mapas de abundancias estimados en un fichero que siga la misma estructura que los ficheros de almacenamiento de imágenes hiperespectrales. Para ello, se deben generar dos ficheros: por un

lado, el fichero binario que contiene la imagen en sí, y que ha de estar ordenado siguiendo una de las tres estructuras explicadas anteriormente; y, por otro, un fichero de cabecera, con extensión *.hdr*, que proporcione los datos necesarios para leer correctamente la imagen, tales como número de píxeles, número de *endmembers* y, en consecuencia, número de mapas de abundancias. La generación de estos dos ficheros se ha implementado en una única función, denominada *gen_hdr*, y crea ambos ficheros con el mismo nombre pero distinta extensión. Estos ficheros se generan en la carpeta *bin*, del proyecto Eclipse, en el mismo directorio donde la compilación crea el ejecutable.

En la figura 3.10 se proporciona el código del actor *Display*; como puede observarse, este actor está compuesto de una única acción, en la que se recibe una matriz (*image_ABUND*) y se generan los ficheros necesarios para la representación de los mapas de abundancias (*gen_hdr*). Además, en esta figura puede observarse la utilización de una función nueva (*end_program*); esta función se utiliza con el objetivo de finalizar la ejecución de la cadena, puesto que la aplicación generada para la cadena no finaliza automáticamente su ejecución. En consecuencia, esta interfaz añade dos funciones nuevas a la librería: *gen_hdr* y *end_program*.

```
receive_abundances: action stream: [token] repeat num_Endmembers ==>

    guard
        received < rc
    do
        array_rec := token;
        foreach int(size= 32)m in 0 .. ((num_Endmembers)-1) do
            image_ABUND[m][received] := float_to_double(array_rec[m]);
        end

        received := received + 1;
        if (received = rc) then
        //  read_txt(ABUND_read, int_Endmembers, rc, "ABUNDANCES1.txt");
            println("Generación de abundancias");
            //write_double_binary(ABUND_read, int_Endmembers, rc, "abund");
            gen_hdr(image_ABUND,int_Endmembers, rc, rows,columns, num_PC, int_Endmembers,
            println("Display: done");
            //write_txt(image_ABUND,int_Endmembers,rc, "ABUNDANCES_DISPLAY.txt");
            println("FIN PROCESAMIENTO");
            end_program();
        end
    end
```

Fig. 3.10. Actor Display

No obstante, además de estas funciones, se han añadido otras, cuya funcionalidad es comprobar el funcionamiento de las diversas etapas de la cadena. Estas funciones son las siguientes:

- *read_txt*: Esta función lee un fichero de texto en el que se ha almacenado una imagen y la guarda en una matriz.
- *write_txt*: Esta función es complementaria de la anterior: escribe una matriz en un fichero de texto. Se ha utilizado para convertir la imagen hiperespectral a analizar en un fichero de texto y comprobar si la interfaz de lectura se ha desarrollado correctamente.
- *write_double_binary*: Esta función escribe una matriz o imagen en un fichero binario, teniendo en cuenta que el tipo de datos de esta matriz es tipo *double*. En otras palabras, su funcionalidad es la misma que la de *gen_hdr*, pero sin generar la cabecera. Esta función ha sido utilizada para comprobar la correcta generación de ficheros binarios.
- *write_short_binary*: Esta función realiza la misma funcionalidad que la anterior, pero cuando el tipo de datos es *short int*.
- *get_time*: Esta función proporciona la hora exacta, de tal forma que permite medir el tiempo total empleado en su ejecución.
- *int_to_double* y *double_to_int*: Facilitan la conversión de tipos *double* a *int*, y viceversa. Estas funciones se han añadido para completar el desarrollo de la librería, puesto que son análogas a las de conversión de tipos *float* y *double*.

Por último, para concluir este apartado se ha de resaltar la ausencia del algoritmo HYSIME en la cadena. Como ya se explicó anteriormente, esto se debe a que la fase de estimación del número de *endmembers* se concibió como una fase opcional, por lo que no se incluye dentro de la cadena de desmezclado como tal. En este Proyecto, dicha fase se ha considerado como una etapa previa de configuración y, por ello, se ha implementado en una red independiente.

Como se explicó en capítulos anteriores, el objetivo de este algoritmo es calcular el número óptimo de *endmembers* de cada imagen hiperespectral, para así optimizar el resto de fases de la cadena de desmezclado. Por tanto, el algoritmo HYSIME proporciona a su salida un valor, que se corresponde con el número de *endmembers* que ha de calcular el actor denominado *VCA*. Además, como se explica en [1], se ha observado que, utilizando el número de *endmembers* proporcionado por HYSIME, la información obtenida para un número de bandas superior a dicho número de *endmembers* es siempre redundante, por lo que se ha establecido que, en el caso óptimo, se han de igualar los parámetros *número de* endmembers y *bandas reducidas*.

El problema que se presenta al contemplar esta etapa como una fase independiente se centra en cómo modificar estos parámetros en la cadena principal de forma autónoma y dinámica. Para ello, se plantean dos soluciones:

- Modificar automáticamente el fichero *Data.cal* al finalizar el algoritmo HYSIME.
- Modificar directamente el código generado por RVC - CAL, cambiando los parámetros necesarios antes de la compilación y generación del ejecutable.

La primera solución hace referencia a un fichero llamado *Data.cal*. Este fichero se muestra en la figura 3.11 y, como puede observarse, contiene variables comunes a todos los actores. Dicho fichero actúa como cualquier paquete en un lenguaje tradicional, por lo que las variables declaradas en él se consideran constantes y, en consecuencia, no pueden ser modificadas de forma dinámica. En concreto, el algoritmo HYSIME necesita modificar tres de sus variables:

- *num_PC*: Variable correspondiente al parámetro aquí denominado *bandas reducidas*.
- *num_Endmembers*: Variable que denota el número de *endmembers* a extraer.
- *int_Endmembers*: Es la misma variable que la anterior, pero almacenada con el tipo de dato *int* (la anterior es *double*). Se realiza esta replicación porque, en determinadas funciones, se requiere el parámetro como *int*, y en otras, como *double*.

Con la primera solución propuesta, lo que haría el algoritmo HYSIME tras calcular el número de *endmembers* sería regenerar el propio fichero *Data.cal*, modificando los valores de las variables mencionadas. Sin embargo, para que estos cambios fueran contemplados en la cadena de desmezclado, habría que regenerar el código desde RVC - CAL y, tras ello, volver a compilar la aplicación. Como puede comprobarse, si bien este proceso no es complicado de realizar, sí requiere la intervención constante de un usuario, ya que no se puede automatizar la generación del código desde RVC - CAL - por ejemplo, mediante un *script*-; esto se traduce en una pérdida de autonomía. Este problema es el que desencadena el desarrollo de la segunda solución, que tiene como objetivo mantener esa autonomía.

Dicha solución consiste en modificar directamente el código autogenerado por RVC - CAL, cambiando las constantes necesarias para establecer los nuevos valores. Este método incrementa la autonomía perdida con la solución anterior, puesto que sólo se necesita generar el código desde RVC - CAL una vez. Por tanto, la ventaja que esta solución introduce es que, en este caso, sí puede utilizarse un *script* para ejecutar la cadena completa - incluyendo HYSIME -; para ello, cabe destacar que debe haberse generado el código desde RVC - CAL en primer lugar. Por su parte, la desventaja de esta solución radica, precisamente, en la modificación del código generado por RVC - CAL, que siempre es un aspecto a evitar.

La implementación de estas dos soluciones ha requerido la adición de dos nuevas funciones a la librería: *gen_conf_data* y *modify_config*. Estas funciones desarrollan la primera y segunda solución, respectivamente:

- Por un lado, *gen_conf_data* crea un nuevo fichero denominado *Data.cal*, idéntico al utilizado en la cadena de desmezclado - mostrado en la figura 3.11 -, pero con las modificaciones necesarias tras la ejecución del algoritmo HYSIME, y lo guarda en el directorio apropiado del proyecto de la cadena, sustituyendo el antiguo.
- Por su parte, *modify_config* busca en todos los archivos con extensión *.c* de dicho proyecto y modifica las variables necesarias para incluir los resultados generados por HYSIME en la cadena de desmezclado. Además, como se explica en el Anexo 3 de este documento, se proporciona un *script* que, automáticamente, compila esta fase previa y, seguidamente, recompila la cadena completa, aumentando así el nivel de autonomía del proceso y minimizando, en consecuencia, la intervención del usuario.

```
package hsi_analysis;


unit Data:


int(size=32) rows = 100;
int(size=32) columns = 100;
int(size=32) bands = 221;
int(size=32) nbits = 16;
int(size=32) nformat = 0;  //0 BSQ, 1 BIP, 2 BIL

int(size=32) num_PC = 10;
double num_Endmembers = 5;

int(size=32) rc = rows*columns;
int(size=32) int_Endmembers = 5;


end
```

Fig. 3.11. Data.cal

Para concluir este apartado, en la tabla 3.9 se recogen las funciones que componen la librería en su totalidad, tal y como se mencionó anteriormente. Como puede comprobarse, esta tabla es, en esencia, similar a la tabla 3.8 mostrada anteriormente, pero se han añadido las funciones explicadas en este apartado, completando así la primera versión de esta librería. Cabe destacar que el total de funciones incluidas ha aumentado de 32 a 45 con la adición de las funciones relativas a interfaz, conversión de tipos de datos y pruebas. Además, estas funciones también pueden considerarse comunes, puesto que tanto este Proyecto Fin de Grado como el desarrollado en [1] requieren, por ejemplo, interfaces de entrada y salida, así como funciones que permitan comprobar el correcto funcionamiento de las propias interfaces.

Por último, cabe destacar que el contenido de la librería se explica en detalle en el Anexo 2 de este documento. Dicho anexo constituye el API de la librería, y en él se realiza una descripción detallada de cada función, explicando su funcionalidad, sus parámetros de entrada y salida y todos los aspectos relevantes para facilitar su utilización.

Asimismo, en el Anexo 3 se proporciona un resumen de los proyectos Eclipse elaborados y proporcionados en este Proyecto Fin de Grado. En dicho anexo, se explica tanto la organización de los directorios como el contenido de los ficheros más importantes, proporcionando la información necesaria para poder utilizar dichos proyectos.

Función	Función
transpose	***get_eigenvectMatrix***
matrix_mult	***matrix_mult_vector***
matrix_scale	***matrix_get_column***
matrix_extend	***matrix_set_column***
matrix_fill_column	***matrix_get_row***
elev	***matrix_set_row***
log	***matrix_reduce***
sq_root	***random_vector***
rem	***getIndexMax***
maxSqrtSum	***get_inverse***
mean_vector	***matrix_plus_value***
vector_minus_vector	***matrix_minus_value***
vector_minus_value	***matrix_minus_matrix***
vector_scale	***matrix_plus_matrix***
get_pinverse	***vector_mult_matrix***
get_diagonal	***vector_mult***
float_to_double	***double_to_float***
int_to_double	***double_to_int***
read_txt	***write_txt***
write_short_binary	***write_double_binary***
modify_config	***gen_conf_data***
gen_hdr	***get_time***
end_program	

Tabla 3 9. Librería completa

División en varios núcleos

Para concluir este capítulo, se va a hacer un breve resumen de las posibilidades de división de la cadena de desmezclado en varios procesadores. Como se ha mencionado a lo largo de este documento, uno de los objetivos de este Proyecto es tratar de aproximarse al tiempo real; se ha comprobado que, para ello, es indispensable ejecutar la aplicación en una plataforma multinúcleo [11]. También se ha explicado que, precisamente, uno de los principales beneficios de la utilización de RVC - CAL es su capacidad de mostrar el paralelismo intrínseco de las aplicaciones desarrolladas y que, además, permite dividir la ejecución de los actores desarrollados en distintos núcleos o procesadores.

En consecuencia, en este apartado se van a analizar las posibles combinaciones existentes a la hora de dividir la ejecución de la cadena en distintos núcleos; tras ello, en el capítulo 4 se probarán las distintas combinaciones explicadas, analizando cuáles son más eficientes respecto al tiempo de ejecución. Para ello, primero se han de seleccionar el número de procesadores en los que se va a ejecutar la aplicación. Los ordenadores utilizados para elaborar este Proyecto son *Intel Core Dúo* con *Ubuntu 12.04*, por lo que sólo se estudiarán las posibles divisiones de actores en dos núcleos.

Teniendo en cuenta, por tanto, que la cadena de procesado de imágenes hiperespectrales desarrollada consta de cinco actores y que, además, éstos van a ser divididos en dos procesadores, el volumen total de distintas combinaciones que pueden extraerse aumenta a 32 (2^5). Para extraer estas combinaciones, se ha simulado una tabla de verdad, en la que un 0 se corresponde con uno de los procesadores, y un 1 con el otro. Analizando el problema desde esta perspectiva, puede observarse que, realmente, de esas 32 combinaciones sólo son útiles 16. Esto se debe a que, al hacer una tabla de verdad, se están contabilizando como dos combinaciones distintas lo que en realidad es la misma combinación; por ejemplo, si existe una combinación en la que el actor *Source* se ejecuta en el procesador 1 y el resto en el procesador 2, también se contabilizará la combinación inversa (*Source* en el procesador 2 y el resto en el procesador 1), cuando en realidad son idénticas. Por ello, en la tabla 3.10 se recogen las combinaciones verdaderamente distintas que pueden obtenerse al dividir cinco actores en dos procesadores. Como puede observarse, la primera de ellas se corresponde con la ejecución en un sistema mononúcleo - todos los actores se ejecutan en un mismo procesador-. Por su parte, las cinco siguientes se corresponden con los casos en que un actor se aísla en un único núcleo y el resto se ejecuta en el otro; por último, las diez combinaciones restantes sitúan dos actores en un procesador y tres en otro.

En el capítulo 4, las 16 combinaciones mostradas en la tabla 3.10 serán ejecutadas, y se proporcionará el tiempo medio de cada una de ellas. Con este dato, se podrán sacar conclusiones acerca de qué combinación es más rápida y cuál es más lenta. Asimismo, este estudio será utilizado y completado en [1], donde, seleccionando la combinación más rápida y la más lenta, se analizará el rendimiento y el consumo de recursos de cada una de ellas, buscando así razones que justifiquen esta diferencia de tiempos entre ambas cadenas.

Actores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Source	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
PCA	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0
VCA	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0
LSU	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1
Display	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1
0 - procesador 1											*1 - procesador 2*					

Tabla 3.10. División en procesadores

CAPÍTULO 4. ANÁLISIS DE RESULTADOS

En el presente capítulo se va a proceder a probar, a través de la cadena de procesado de imágenes hiperespectrales, la librería desarrollada en este Proyecto Fin de Grado. En este apartado se analizará tanto la bondad de los algoritmos implementados - y, en consecuencia, de la librería - como parámetros relacionados con la eficiencia, el rendimiento y la velocidad de ejecución. Asimismo, se estudiará la variación de los tiempos de ejecución de la cadena para distintas distribuciones en procesadores.

En concreto, la organización de este capítulo es la siguiente: en primer lugar, se van a probar de forma individual las fases desarrolladas en este trabajo (estimación del número de *endmembers* y estimación de las abundancias), analizando tanto la bondad de los algoritmos desarrollados como el tiempo de ejecución medio de cada uno de ellos. En segundo lugar, se procederá a estudiar la cadena completa; para ello, se tendrán en cuenta los resultados extraídos por Daniel Madroñal en [1] sobre las otras dos fases de la cadena. En concreto, la segunda parte de este documento se centrará en analizar el funcionamiento de la cadena completa cuando ésta se implementa en varios actores. Este análisis partirá, nuevamente, de [1], donde se realiza el mismo estudio pero con la cadena implementada en un único actor; de esta forma, se podrán extraer resultados exactos acerca del efecto del error de comunicación en el proceso de análisis de la imagen. Además, este análisis se realizará tanto en un procesador como en varios; así, se podrá observar el impacto de la división de procesadores en la ejecución de la aplicación. Por último, este capítulo concluirá con el análisis de los resultados obtenidos tras aplicar la herramienta PAPI - y, en concreto, el software *papify*[18] desarrollado por el compañero Alejo Arias - a la cadena completa de desmezclado espectral.

Como ya se mencionó en el capítulo anterior, para comprobar la bondad de la cadena completa, en general, y de los algoritmos, en particular, se va a hacer uso de *Hypermix*, que es una herramienta software de procesado de imágenes hiperespectrales desarrollada por la Universidad de Extremadura [14]. Las principales ventajas de esta herramienta son, por un lado, que es software libre, y, por otro, que proporciona un visor de imágenes hiperespectrales, *endmembers* y abundancias. Estas características han hecho que su utilización en este Proyecto sea de gran utilidad: por un lado, el visor de dicha herramienta será utilizado para comprobar la bondad de los algoritmos desarrollados, comparando tanto los mapas de abundancias como el número de *endmembers* estimado por dicho programa con los obtenidos en este Proyecto para una serie de imágenes; por otro lado, la naturaleza libre de este software permite la instrumentación del mismo, de tal forma que puedan añadirse las funciones necesarias para medir el tiempo de ejecución de los algoritmos en dicho software y, así, poder compararlo con el obtenido en los algoritmos desarrollados en este Proyecto.

Por último, conviene destacar que las imágenes utilizadas en este trabajo para la evaluación del sistema construido han sido extraídas de la página oficial de *Hypermix*[19]. Estas imágenes, denominadas *fractals*, son sintéticas, esto es, han sido construidas de forma artificial, basándose en un conjunto de firmas espectrales extraídas de la base de datos de la USGS - *United States Geological Survey* -. En total, se aportan cinco imágenes distintas y, de cada una de ellas, se proporcionan varias versiones: seis versiones con distintos niveles de ruido, y una versión sin ruido añadido. Además, como se mencionó anteriormente, estas imágenes contienen un total de 10000 píxeles, con una resolución espectral de 221 bandas.

[18] https://github.com/alejoar/papify
[19] http://www.hypercomp.es/hypermix/Downloads

4.1. Análisis por fases

Como se acaba de mencionar, en este apartado se explicarán las pruebas realizadas para comprobar tanto la bondad como la velocidad de los algoritmos desarrollados para las fases de estimación del número de *endmembers* y estimación de las abundancias. Este mismo análisis, pero para las fases de reducción dimensional y extracción de *endmembers*, está detalladamente explicado en el Proyecto Fin de Grado de Daniel Madroñal [1]; los resultados extraídos en él serán utilizados en el siguiente apartado para evaluar la cadena completa.

Por tanto, en primer lugar se analizarán la bondad y los tiempos de ejecución del algoritmo HYSIME. Para analizar la bondad, se realizarán dos tipos de pruebas:

- Inicialmente, se ejecutará el algoritmo HYSIME para las cinco *fractals* proporcionadas por *Hypermix*. Dicho algoritmo será ejecutado tanto en el sistema desarrollado en este Proyecto Fin de Grado como en la propia herramienta software mencionada; la bondad del algoritmo podrá considerarse comprobada si los resultados obtenidos en ambas pruebas son idénticos.
- Tras ello, se ejecutará dicho algoritmo sobre alguna de las imágenes con ruido que *Hypermix* proporciona. El objetivo de esta prueba es comprobar el grado de robustez del algoritmo frente a la presencia de ruido en la imagen. No obstante, esto es un objetivo secundario para completar el análisis por fases, puesto que el objetivo principal de la implementación del algoritmo es comprobar que éste presenta un comportamiento idéntico al desarrollado en *Hypermix* y que, en consecuencia, las funciones desarrolladas para la librería funcionan correctamente.

Por su parte, para el análisis de tiempos ha sido necesario instrumentar el código del software *Hypermix* para que proporcione el tiempo de ejecución del algoritmo. Esto se ha realizado utilizando un planteamiento similar al de la función *get_time* desarrollada en la librería de este Proyecto Fin de Grado; asimismo, dicha función ha sido utilizada en el sistema desarrollado para obtener el tiempo de ejecución del algoritmo al utilizar dicha librería. En ambos casos, cabe destacar que el tiempo que se ha medido ha sido únicamente el del algoritmo, sin tener en cuenta el tiempo consumido al leer la imagen o al escribir el resultado.

Para familiarizarse con las imágenes que se van a utilizar a lo largo de este capítulo, en la tabla 4.1 se recoge una muestra de cada una de ellas. Como puede observarse, estas imágenes se reducen a cinco escenas distintas. De estas cinco escenas se proporcionan siete imágenes: una de ellas, sin ruido añadido, y las seis siguientes con distintas relaciones señal a ruido (SNR): 110, 90, 70, 30 y 10. En dicha tabla puede apreciarse que la primera columna muestra las imágenes más nítidas - sin ruido añadido -, mientras que la última muestra las imágenes menos nítidas, en las que apenas puede apreciarse nada - relación señal a ruido muy baja.

Respecto a las pruebas que se van a realizar a lo largo del presente capítulo, conviene destacar que los resultados que aquí se proporcionan se han obtenido tras repetir cada prueba un determinado número de veces y hacer la media de los resultados obtenidos en cada una de las repeticiones. Este será el patrón de ejecución de pruebas, independientemente de si las pruebas realizadas son de bondad o de tiempo de ejecución; no obstante, estas repeticiones serán especialmente importantes en las pruebas de tiempos, puesto que la precisión de esas medidas es mucho mayor que, por ejemplo, la precisión de los resultados del algoritmo HYSIME, como se verá a continuación. Por ello, dichas pruebas se repetirán un mínimo de diez veces, mientras que las de bondad sólo requerirán cinco repeticiones.

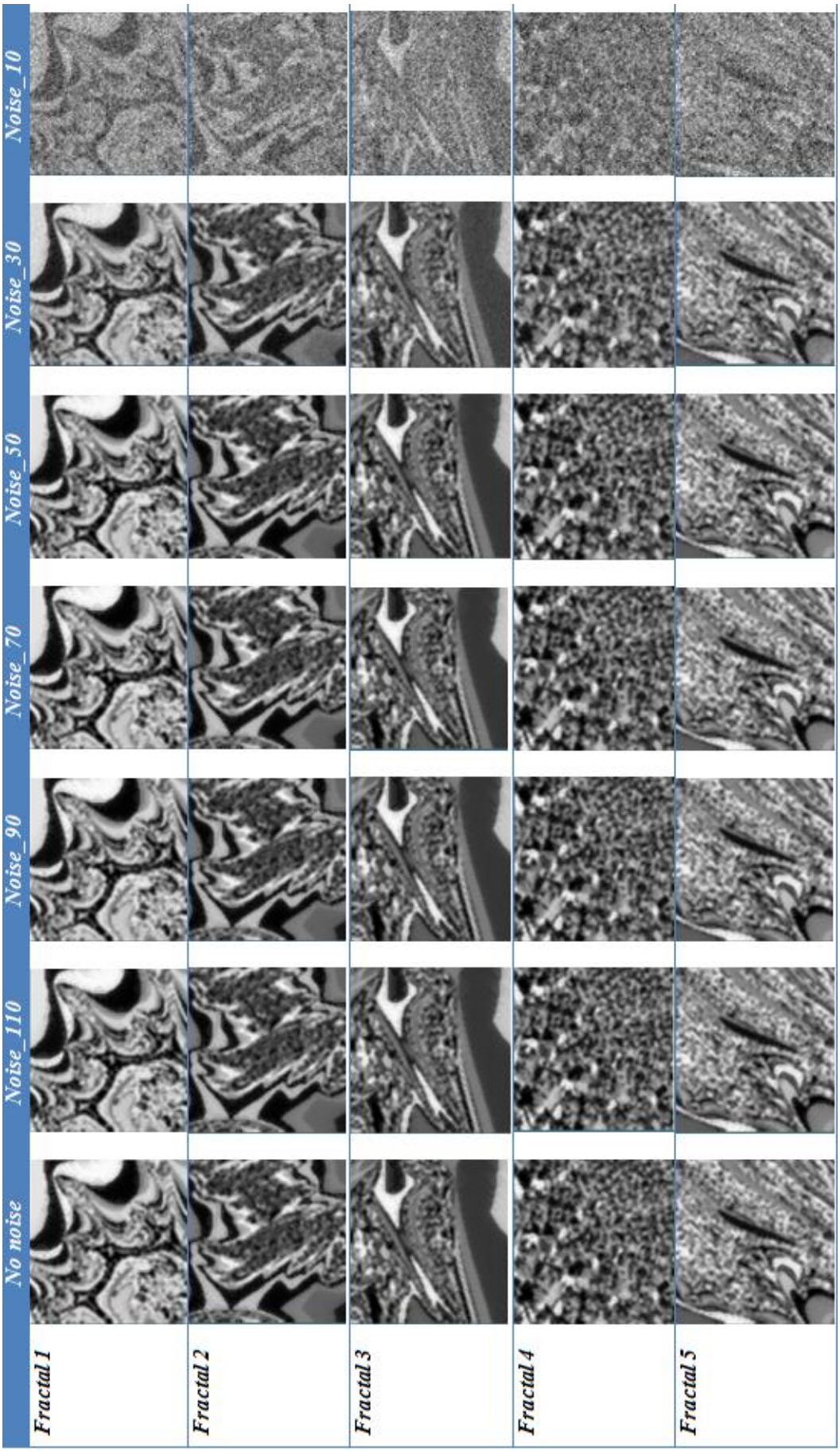


Tabla 4.1. Imágenes utilizadas (Fuente: Universidad de Extremadura)

Tras ejecutar el mencionado algoritmo tanto en *Hypermix* como en el sistema desarrollado en RVC - CAL, los resultados obtenidos para las cinco *fractals* sin ruido añadido se recogen en la tabla 4.2. Cabe mencionar que, como se ha mencionado antes, estas pruebas se han repetido diez veces y, para cada una de las imágenes, los resultados obtenidos siempre han sido los mismos.

Como puede observarse en dicha tabla, tanto los resultados obtenidos mediante la herramienta *Hypermix* como los obtenidos en este Proyecto Fin de Grado coinciden completamente. Como ya se adelantaba anteriormente, esto demuestra la bondad del algoritmo implementado en RVC - CAL y, en consecuencia, la bondad de las funciones implementadas en la librería para la construcción de este algoritmo.

Fractal	Resultado *Hypermix*	Resultado Proyecto Fin de Grado
Fractal 1	6	6
Fractal 2	6	6
Fractal 3	6	6
Fractal 4	6	6
Fractal 5	5	5

Tabla 4.2. Resultados de Hypermix *para las imágenes sin ruido*

Para completar el estudio de la bondad del algoritmo, en la tabla 4.3 se proporcionan los resultados obtenidos para las cinco *fractals* con una relación señal a ruido de 110. Como puede observarse, para las *fractals* 1,2 y 5 los resultados obtenidos en ambos casos coinciden con los de las imágenes sin ruido; sin embargo, para las imágenes 3 y 4 ya se introduce un ligero error, puesto que el ruido presente en la imagen hace que el algoritmo detecte un *endmember* de más. De esto puede concluirse, de forma genérica, que el algoritmo HYSIME presenta un comportamiento robusto frente al ruido hasta cierto punto, puesto que, cuando el ruido presente en la imagen supera un determinado umbral, el resultado del cálculo de *endmembers* deja de coincidir con el de la misma imagen, pero sin ruido. Para comprobar esta hipótesis, se ha ejecutado el algoritmo en *Hypermix* para las imágenes con una relación señal a ruido de 90 y, efectivamente, se ha comprobado que, en esta ocasión, ninguno de los resultados obtenidos coincide con los aquí mostrados, por lo que se puede afirmar que la presencia de ruido en la imagen a analizar puede introducir error en el cálculo.

Fractal	Resultado *Hypermix*	Resultado Proyecto Fin de Grado
Fractal 1	6	6
Fractal 2	6	6
Fractal 3	7	7
Fractal 4	7	7
Fractal 5	5	5

Tabla 4.3. Resultados de Hypermix *para las imágenes con SNR = 110*

Una vez completado el estudio de la bondad del algoritmo y, en consecuencia, de las funciones de la librería desarrolladas para su implementación, se va a proceder al análisis del tiempo computacional asociado a este algoritmo. Para ello, se ha escogido una de las imágenes utilizadas anteriormente - en este caso, la *fractal* 2 en su versión sin ruido - y se le ha aplicado dicho algoritmo - tanto en *Hypermix* como en el software desarrollado en este trabajo - un total de diez veces. Los resultados que se han obtenido en ambos casos para cada una de las pruebas, así como la media, el valor mínimo y el valor máximo, aparecen recogidos en la tabla 4.4.

Como puede observarse en dicha tabla, los resultados obtenidos para la cadena implementada en RVC - CAL son sustancialmente mejores que los obtenidos mediante la herramienta *Hypermix*. De forma genérica, podemos apreciar que el tiempo de ejecución del algoritmo HYSIME en *Hyermix* varía entre 26.86 s y los 27.16 s, aproximadamente; por su parte, utilizando las funciones añadidas a la librería, este algoritmo presenta un tiempo de ejecución comprendido entre 19.51 s y 19.59 s, también de forma aproximada.

En concreto, atendiendo a la media de las diez medidas realizadas para cada uno de los casos, puede comprobarse que la implementación del algoritmo mediante la aplicación de la librería desarrollada en este Proyecto Fin de Grado supone una aceleración del tiempo de ejecución de un 27.61% respecto al tiempo de ejecución del algoritmo en *Hypermix*. Este resultado es especialmente relevante, puesto que la sustancial reducción de tiempo en la ejecución del algoritmo con respecto a *Hypermix* parece confirmar la utilidad de la construcción de una librería de análisis de imágenes hiperespectrales, ahora desde el punto de vista del tiempo de ejecución.

Pruebas	**Tiempos *Hypermix***	**Tiempos Proyecto Fin de Grado**
Prueba 1	27.1605	19.516652
Prueba 2	27.0257	19.587679
Prueba 3	26.9915	19.574518
Prueba 4	26.96	19.552871
Prueba 5	27.0914	19.506122
Prueba 6	27.0576	19.576645
Prueba 7	26.898	19.519879
Prueba 8	26.8586	19.551259
Prueba 9	26.8656	19.520745
Prueba 10	27.1207	19.568434
Media	27.00296	19.5474804
Máximo	27.1605	19.587679
Mínimo	26.8586	19.506122

Tabla 4.4. Resumen análisis de tiempos (en segundos) del algoritmo HYSIME

En consecuencia, de este análisis se puede concluir que el algoritmo desarrollado para la fase de estimación de *endmembers* obtiene los mismos resultados que la herramienta *Hypermix*, pero en un tiempo sustancialmente menor, lo que pone de manifiesto, en primer lugar, la utilidad de la librería desarrollada y, en segundo lugar, su potencial para alcanzar el tiempo real. No obstante, de los tiempos de ejecución obtenidos puede concluirse que, para poder acercarse a dicho objetivo, este algoritmo necesita ser optimizado.

Una vez analizada tanto la bondad como el tiempo de ejecución de la fase de estimación de *endmembers*, se va a proceder a realizar el mismo análisis para la fase de estimación de las abundancias - algoritmo *LSU* -. Conviene destacar que, para analizar esta fase, se han tenido en cuenta los resultados analizados por Daniel Madroñal en [1] para las fases de reducción dimensional y extracción de *endmembers*. Por ello, se recomienda consultar primero dicho análisis antes de continuar la lectura del presente documento.

Como menciona Daniel Madroñal en [1], la fase de extracción de *endmembers* consta de una componente aleatoria que hace que los resultados extraídos por la herramienta desarrollada en este Proyecto Fin de Grado y los obtenidos mediante *Hypermix* puedan presentar ligeras variaciones. Teniendo en cuenta que la fase que se va a analizar ahora se apoya en las dos etapas explicadas en [1], para comprobar la bondad de esta última fase de la cadena de desmezclado espectral se va a seguir el mismo planteamiento que para la fase de extracción de *endmembers*:

- En primer lugar, se realizará un breve análisis de esta última fase sin tener en cuenta la componente aleatoria antes mencionada. Para ello, se tomará como entrada al algoritmo *LSU* el conjunto de *endmembers* calculados en [1] para el caso en el que la componente aleatoria es suprimida y sustituida por los valores generados por *Hypermix*. De esta forma, se consigue comprobar de forma fiable la bondad del algoritmo, puesto que, al suprimir dicha componente aleatoria, los resultados obtenidos tanto en *Hypermix* como en la cadena desarrollada en este Proyecto Fin de Grado deben ser idénticos.
- Tras ello, se procederá a comprobar el efecto de dicha aleatoriedad en el cálculo de abundancias. Este análisis servirá, asimismo, para corroborar la hipótesis planteada por Daniel Madroñal [1], en la que afirma que el efecto de la componente aleatoria sobre el conjunto de *endmembers* calculado supone un desorden en el mismo. Si esto fuera cierto, el efecto que provocaría a la salida de la cadena sería que, respecto a *Hypermix*, los mapas de abundancias generados estarían también desordenados.
- Una vez analizada la bondad del algoritmo, se completará el estudio con el análisis del tiempo de ejecución en ambas herramientas, de forma equivalente a como se ha realizado en la fase anterior.

Por tanto, la primera prueba realizada ha sido la de ejecutar el algoritmo *LSU* tanto en *Hypermix* como en RVC - CAL, pero tomando como entrada los *endmembers* generados en [1] para el caso de no aleatoriedad. Estas pruebas se han realizado con dos imágenes:

- *Fractal* 1 reducida a 100 bandas, de la que se han extraído dos conjuntos distintos de *endmembers* - 15 y 20, respectivamente -. Además, para la segunda parte explicada - comprobación del efecto de la aleatoriedad en las abundancias - también se ha utilizado esta misma imagen, pero reducida a 25 bandas y con un conjunto de 10 *endmembers*.
- *Fractal* 2 reducida a 25 bandas, de la que también se han extraído dos conjuntos de *endmembers* - 5 y 10, respectivamente.

En ambos casos, para facilitar el proceso de análisis se ha hecho uso de la herramienta *Meld Diff Viewer*[20]. Esta herramienta permite, entre otras funcionalidades, comparar rápidamente dos ficheros de texto. Por tanto, para poder utilizar esta herramienta, se ha hecho uso de las funciones *read_txt* y *write_txt* para escribir el conjunto de *endmembers* en un fichero de texto - al final de la etapa de extracción de *endmembers* - y para leer el mismo al inicio de la fase de estimación de abundancias. Asimismo, también se ha instrumentado *Hypermix*, de tal forma que guarde las abundancias calculadas en un fichero de texto.

Tras la realización de estos cambios, se ha ejecutado el algoritmo para los cuatro conjuntos de *endmembers* disponibles y, con el *Meld Diff Viewer*, se ha comprobado que, en todos los casos, los resultados extraídos de *Hypermix* y de RVC - CAL son idénticos. Además, para comprobar esta igualdad en los resultados obtenidos, se ha utilizado el visor de *Hypermix*: abriendo ambos mapas de abundancias simultáneamente, se ha comprobado que, para cada una de las pruebas realizadas, estos mapas son idénticos.

[20] http://meldmerge.org/

En la tabla 4.5 se proporciona, a modo ilustrativo, los resultados obtenidos en este visor para el caso de la *fractal* 2, reducida a 25 bandas y con 5 *endmembers*. La fila de arriba se corresponde con los mapas de abundancia calculados por *Hypermix*, mientras que los de la fila de abajo se corresponden con los calculados en este Proyecto Fin de Grado, mediante la aplicación de la librería desarrollada. Como se puede observar, los mapas de abundancias calculados mediante ambas herramientas son idénticos, tanto en orden como en contenido. Esto demuestra que el algoritmo implementado en RVC - CAL y, en consecuencia, las funciones desarrolladas en la librería para el mismo, funcionan correctamente; por ello, la bondad del algoritmo implementado queda demostrada.

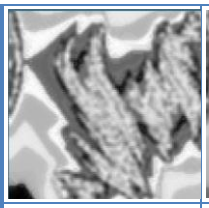
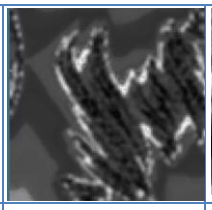
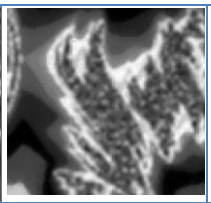
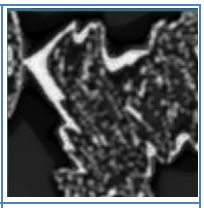
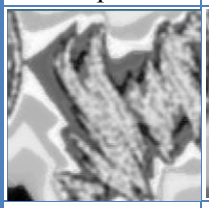
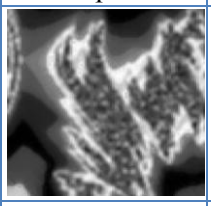
Hyper-mix					
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5
RVC - CAL					
	Mapa 6	Mapa 7	Mapa 8	Mapa 9	Mapa 10

Tabla 4.5. Comparativa mapas de abundancia de la fractal *2 para 5* endmembers

A continuación, una vez que se ha probado la bondad del algoritmo sin aleatoriedad, se va a estudiar cuál es el efecto de la misma en el cálculo de abundancias. Para ello, se han utilizado las imágenes mencionadas anteriormente y, para los distintos conjuntos de *endmembers* proporcionados, se han calculado las abundancias en una y otra herramienta. Por último, utilizando el visor de *Hypermix*, se han comparado los resultados obtenidos. En la tabla 4.6 se recoge dicha comparativa para el caso de la *fractal* 1, reducida a 25 bandas y con 10 *endmembers*. De dicha tabla, la principal conclusión que puede extraerse es que los mapas de abundancia calculados por *Hypermix* y por RVC - CAL también coinciden, pero aparecen desordenados. Así, se pueden establecer las siguientes correspondencias:

- El mapa 1 estimado por la herramienta *Hypermix* se corresponde con el mapa 2 calculado por RVC - CAL.
- El mapa 2 estimado por la herramienta *Hypermix* se corresponde con el mapa 1 calculado por RVC - CAL.
- Los mapas 3, 9 y 10 calculados por ambas herramientas coinciden entre sí.
- El mapa 4 estimado por la herramienta *Hypermix* se corresponde con el mapa 8 calculado por RVC - CAL.
- El mapa 5 estimado por la herramienta *Hypermix* se corresponde con el mapa 7 calculado por RVC - CAL.
- El mapa 6 estimado por la herramienta *Hypermix* se corresponde con el mapa 5 calculado por RVC - CAL.
- El mapa 7 estimado por la herramienta *Hypermix* se corresponde con el mapa 6 calculado por RVC - CAL.
- El mapa 8 estimado por la herramienta *Hypermix* se corresponde con el mapa 4 calculado por RVC - CAL.

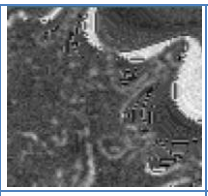
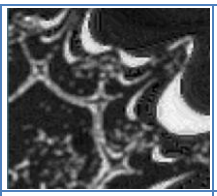
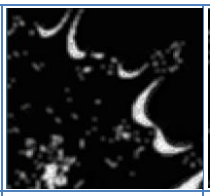
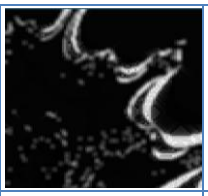
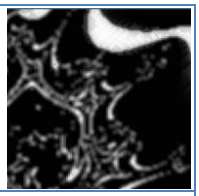
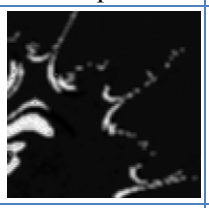
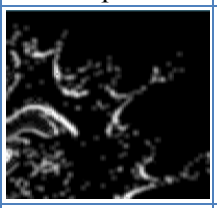
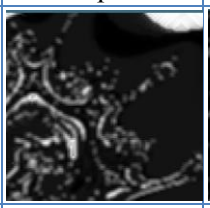
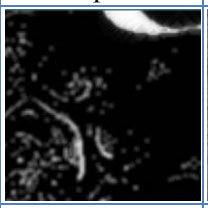
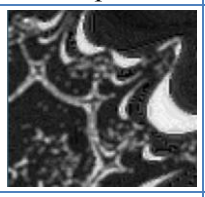
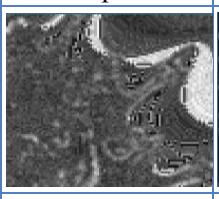
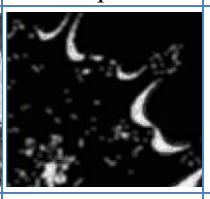
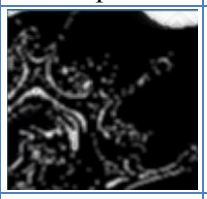
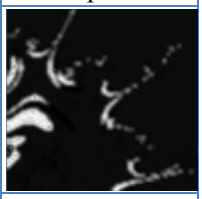
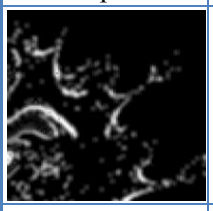
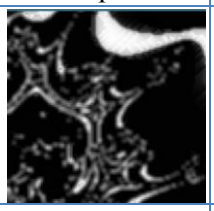
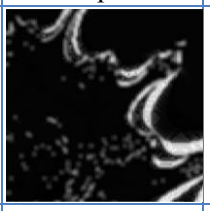
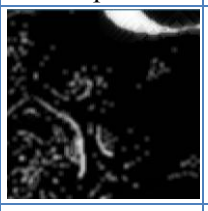
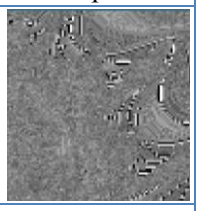
Hyper-mix					
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5
					
	Mapa 6	Mapa 7	Mapa 8	Mapa 9	Mapa 10
RVC - CAL					
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5
					
	Mapa 6	Mapa 7	Mapa 8	Mapa 9	Mapa 10

Tabla 4.6. Comparativa mapas de abundancia de la fractal *1 para 10* endmembers

De las relaciones expuestas, se puede concluir que, tal y como afirmaba Daniel Madroñal en [1], efectivamente el efecto de la aleatoriedad de la fase de extracción de *endmembers* en la estimación de abundancias no es otra que un desorden en los mapas de abundancia. Además, tal y como se puede deducir de las relaciones, es evidente que todos los mapas de abundancia calculados por *Hypermix* se corresponden con los calculados por RVC - CAL; en otras palabras, no existe ningún mapa de abundancia que no esté presente en ambos casos. En consecuencia, queda suficientemente probada la hipótesis de que el efecto de la aleatoriedad se traduce generalmente en una variación en la ordenación de los mapas de abundancia.

Por último, para concluir este apartado se va a proceder a hacer un análisis de los tiempos de ejecución del algoritmo *LSU*, realizando una comparativa entre el tiempo de *Hypermix* y el de RVC - CAL. Para ello, siguiendo con la metodología expuesta tanto en la primera fase del presente documento como en las dos fases explicadas en [1], se ha utilizado la *fractal* 2 reducida tanto a 25 como a 100 bandas, y de cada uno de los casos se ha ejecutado dicho algoritmo cuando el conjunto de *endmembers* contiene 5 y 20 *endmembers*, respectivamente. De esta forma, se cubren todas las combinaciones extremas, ya que, para los casos mínimo y máximo de bandas reducidas, se calculan los casos mínimo y máximo de número de *endmembers*. Así, se proporcionan los casos más simple y más complejo - 25 bandas, 5 *endmembers* y 100 bandas, 20 *endmembers*, respectivamente - y dos casos intermedios. Asimismo, cabe destacar que, tal y como se remarcaba al inicio del capítulo, cada prueba se ha repetido un mínimo de diez veces. En este caso, en la tabla 4.7 se proporcionan los resultados para estas pruebas, recogiéndose únicamente los valores mínimo, medio y máximo de cada una de las mismas.

De la observación de dicha tabla pueden obtenerse varias conclusiones relevantes:

- En primer lugar, lo primero que se puede apreciar es que, para todos los casos, los valores de tiempo obtenidos con la cadena desarrollada en RVC - CAL son sustancialmente inferiores a los desarrollados en *Hypermix*. En concreto, el porcentaje de mejora conseguido en cada caso figura en la última columna de la tabla; como puede observarse, este porcentaje llega a variar desde un 17.46% hasta un 62.04%. Uniendo este resultado con los obtenidos en el algoritmo HYSIME, parece comprobarse que la aplicación en RVC - CAL de la librería desarrollada consigue, de forma genérica, una importante reducción en el tiempo de ejecución en los algoritmos de la cadena de análisis de imágenes hiperespectrales. Esta observación es especialmente importante, puesto que acerca el objetivo del tiempo real.
- En segundo lugar, otro aspecto que resulta relevante y que puede extraerse directamente de la tabla es el hecho de que, para todos los casos, los resultados proporcionados por RVC - CAL proporcionan valores más constantes que los proporcionados por *Hypermix*. Esto puede observarse, por ejemplo, en el caso de los resultados obtenidos para la imagen reducida a 25 bandas y 5 *endmembers*; como puede comprobarse, la diferencia máxima entre el tiempo mínimo y máximo para *Hypermix* es de 0.011, aproximadamente, mientras que la misma diferencia para RVC - CAL es de apenas 0.00019. La principal ventaja que esto aporta es que, para una misma plataforma y en las mismas condiciones, el tiempo de ejecución de este algoritmo es prácticamente constante, lo que facilita el modelado del comportamiento de dicho algoritmo y, en consecuencia, contribuye a la planificación y distribución eficiente de los distintos núcleos de la plataforma.

Pruebas			***HyperMix***	**RVC – CAL**	**Mejora**
25 bandas	5 *endmembers*	Mínimo	0.0153213	0.009771	36.23%
		Media	0.01788953	0.0098678	44.84%
		Máximo	0.0262415	0.009961	62.04%
	20 *endmembers*	Mínimo	0.0596848	0.037544	37.10%
		Media	0.06457155	0.040275	37.63%
		Máximo	0.0702948	0.041328	41.21%
100 bandas	5 *endmembers*	Mínimo	0.0618043	0.044802	27.51%
		Media	0.06481092	0.0449143	30.70%
		Máximo	0.0694778	0.045195	34.95%
	20 *endmembers*	Mínimo	0.218299	0.179971	17.56%
		Media	0.2394821	0.180406	24.67%
		Máximo	0.313818	0.181416	42.19%

Tabla 4.7. Comparativa de tiempos (en segundos) del algoritmo LSU

En conclusión, se ha comprobado que, al igual que ocurría con la fase de estimación del número de *endmembers*, el algoritmo desarrollado para la etapa de estimación de abundancias no sólo proporciona resultados equivalentes a los generados por la herramienta *Hypermix*, sino que, además, consigue una reducción de tiempo sustancial en la ejecución del mencionado algoritmo, acercando así el objetivo de tiempo real.

4.2. Análisis de la cadena completa

En el apartado anterior se analizó la bondad y los tiempos de ejecución de las dos fases de la cadena de desmezclado espectral estudiadas en este Proyecto Fin de Grado; además, en [1] se realizó un estudio análogo para las dos fases restantes de dicha cadena. Las conclusiones extraídas en ambos documentos son idénticas: los algoritmos implementados en RVC - CAL presentan un comportamiento equivalente a los de *Hypermix* - desde el punto de vista de la bondad - y, además, consiguen una reducción sustancial en el tiempo de ejecución.

Estas conclusiones permiten avanzar a la siguiente fase de este Proyecto Fin de Grado. Como se explicó en el apartado anterior, tras implementar las dos fases por separado, se realizó la unificación de la librería y, tras ello, se procedió a implementar la cadena completa de procesado de una imagen hiperespectral en RVC - CAL. De esta cadena se construyeron dos configuraciones distintas:

- En primer lugar, se implementó la cadena completa únicamente en un actor. El objetivo de esta implementación es, por un lado, detectar posibles problemas a la hora de enlazar las fases y, por otro, proporcionar una cadena en la que se pueda medir el tiempo de ejecución de la misma sin que afecten los problemas de comunicación entre actores explicados en el capítulo anterior. Asimismo, también servirá para probar la bondad de la cadena completa, comprobando que los resultados de la misma son equivalentes a los obtenidos en el apartado anterior tanto de este documento como de [1]. Este análisis se ha realizado en profundidad en el trabajo de Daniel Madroñal [1], por lo que será citado asiduamente a lo largo de este epígrafe.
- En segundo lugar, se implementó la cadena explicada en el capítulo anterior. Como se expuso en dicho capítulo, esta cadena está compuesta de cinco actores: los tres correspondientes a las fases obligatorias de la cadena - reducción dimensional, extracción de *endmembers* y estimación de abundancias - y los correspondientes a la interfaz de entrada y de salida. El objetivo de implementar esta cadena es analizar el efecto que los errores de comunicación introducen en la cadena de desmezclado espectral, comparando los resultados obtenidos en este caso con los extraídos de la cadena implementada en un único actor. Asimismo, otro de los objetivos para implementar esta cadena es estudiar el impacto que produce la división en procesadores - esto es, la ejecución en paralelo - en el tiempo de ejecución de la cadena completa.

En consecuencia, la organización de este capítulo es la siguiente:

- En primer lugar, se analizará la influencia de los errores de comunicación en la cadena completa. Para ello, se partirá del análisis realizado en [1] para la cadena completa en un único actor. Al igual que se realizó en el anterior apartado, en éste se analizará, inicialmente, la cadena completa, pero eliminando el efecto de la aleatoriedad; de esta forma, como ya se demostró la bondad de los algoritmos por separado, la variación que exista entre los resultados extraídos en [1] y los obtenidos para la configuración en varios actores se deberá exclusivamente al error de comunicación añadido. De este modo, podrá analizarse el efecto de este error de forma aislada, sin que intervengan otros factores como la aleatoriedad. Para realizar este apartado, se utilizará la *fractal* 1.
- En segundo lugar, se comprobará que el efecto de la aleatoriedad en la cadena completa sigue siendo el mismo que en los algoritmos individuales: el desorden del conjunto de *endmembers* y, en consecuencia, el desorden en los mapas de abundancia. Para realizar este apartado, se utilizará la *fractal* 2.

- Por último, se realizará el análisis de tiempos. Este estudio es el más importante del presente apartado, puesto que, en él, se ejecutarán las distintas divisiones en procesadores diseñadas en el capítulo anterior y se obtendrán los tiempos de ejecución de cada una de ellas. Esto permitirá realizar una comparativa entre las distintas divisiones, observando cuáles son más rápidas y cuáles son más lentas. Asimismo, también permitirá hacer la comparativa entre la ejecución en un solo procesador frente a la ejecución en varios, observando los efectos que esto ocasiona.

Por tanto, una vez explicada la organización de este apartado, se va a proceder al análisis de la cadena completa, en 5 actores y sin aleatoriedad. Las pruebas que se han realizado para esta comprobación se pueden dividir en dos bloques: por un lado, se ha estudiado un caso denominado estándar, que aplica la reducción PCA a 25 bandas y calcula 15 *endmembers* para la *fractal* 1; y, por otro, se ha analizado el denominado caso óptimo, que aplica los parámetros calculados por el algoritmo HYSIME a la misma imagen y, en consecuencia, la imagen se reduce a 6 bandas y se calculan únicamente 6 *endmembers*.

Al igual que para el caso de la estimación de abundancias, para analizar esta prueba se ha utilizado el software *Meld Diff Viewer* que, como se mencionó en el apartado anterior, permite comparar ficheros de texto. Este programa se ha utilizado para comparar la salida de cada actor de la cadena - más concretamente, la salida de los actores *PCA, VCA* y *LSU* - con las mismas salidas, pero de la cadena implementada únicamente en un actor, de la que se analizan los resultados en [1].

Sin embargo, los resultados que este software ha arrojado no han sido satisfactorios: tal y como se esperaba, los ficheros generados por una y otra configuración no son idénticos. Esto se debe exclusivamente al error añadido en la comunicación - detallado en el capítulo anterior -; como se explicaba anteriormente, este error se debe a la transformación de tipo de datos que debe hacerse tanto para el envío como para la recepción de datos - de *double* a *float* y viceversa -, que redondea las cifras y, consecuentemente, introduce variaciones en los decimales de los datos. Conviene destacar que este error es lo único que puede hacer variar la salida de esta cadena con respecto a la analizada en [1], por lo que, si se cuantifica la diferencia entre uno y otro fichero, se podrá estimar el efecto del error de comunicación en el análisis de la imagen.

Para medir el grado de influencia de este error en el resultado final, se ha procedido a calcular el error cuadrático medio entre los resultados generados por la cadena implementada en un actor y la implementada en cinco. De esta forma, se podrá aportar un análisis cuantitativo del error introducido por la comunicación al implementar la cadena en varios actores. En consecuencia, se ha calculado el error cuadrático medio para tres resultados distintos:

- En primer lugar, se ha calculado el error cuadrático medio para la salida generada por el actor *PCA*; esta salida es una matriz que contiene la imagen reducida espectralmente al número de bandas especificado.
- En segundo lugar, se ha calculado el error cuadrático medio para la salida generada por el actor *VCA*; esta salida es una matriz que contiene el conjunto de *endmembers* calculados para la imagen reducida en el actor *PCA*.
- Por último, se ha calculado el error cuadrático medio para la salida total de la cadena, que es la generada por el algoritmo *LSU*; al igual que en las anteriores, esta salida también es una matriz, pero en esta ocasión contiene las abundancias estimadas para los *endmembers* extraídos en el actor *VCA*.

Por su parte, el error cuadrático medio se ha calculado de la siguiente forma:

- Inicialmente, se restan, elemento a elemento, las matrices obtenidas como resultado de ambas configuraciones - por ejemplo, la matriz obtenida en el algoritmo PCA de la cadena completa en un actor y la matriz análoga obtenida por el actor *PCA* de la cadena de 5 actores.
- En la matriz resultante, se eleva cada elemento al cuadrado y, posteriormente, se suman todos los elementos resultantes. En consecuencia, tras este paso el resultado debe ser un único número.
- Por último, se divide el resultado obtenido en el paso anterior entre el número total de elementos existentes en la matriz. El resultado obtenido es el error cuadrático medio.

En la tabla 4.8 se recoge el valor calculado para cada uno de los casos explicados. Como puede observarse, en todos los casos el error es inferior a la millonésima parte, por lo que, a simple vista, parece que el error de comunicación puede considerarse despreciable a efectos prácticos. Sin embargo, conviene realizar esa comprobación directamente sobre los mapas de abundancia, de forma similar a como se realizó en el apartado anterior entre *Hypermix* y RVC - CAL.

Por ello, se utilizará el visor de *Hypermix* para visualizar simultáneamente los mapas de abundancias estimados por la cadena de un actor y los generados por la de varios actores; de esa forma, podrá comprobarse cuál es el efecto visual real de este error en el análisis de la imagen.

Pruebas	Caso estándar	Caso óptimo
Salida PCA	$< 10^{-6}$	$< 10^{-6}$
Salida VCA	$< 10^{-6}$	$< 10^{-6}$
Salida LSU	$< 10^{-6}$	$< 10^{-6}$

Tabla 4.8. Cálculo del error cuadrático medio

Como en el caso estándar se estiman un total de 15 mapas de abundancia, el total de imágenes a comparar aumentaría a 30, lo que supone un esfuerzo innecesario para el objetivo de dicha comparación. Por ello, en este documento sólo se proporcionan y explican los mapas de abundancias obtenidos en ambos casos para el caso óptimo - 6 *endmembers*.

En consecuencia, en la tabla 4.9 se recogen los mapas de abundancia extraídos para el caso óptimo mediante las dos configuraciones desarrolladas. Como puede observarse, resulta evidente que, al no haber aleatoriedad, los mapas de abundancia de ambos casos aparecen ordenados: el primer mapa de la cadena en un actor se corresponde con el primer mapa para la cadena en 5 actores, y lo mismo ocurre para los sucesivos.

Además, como se indica en la tabla, las imágenes proporcionadas para la cadena en 5 actores contienen el ruido de comunicación; si se comparan dichos mapas de abundancia con los obtenidos en el caso de la cadena en un único actor, puede comprobarse que, efectivamente, a simple vista no se aprecia ninguna diferencia.

Cabe destacar que, como ya se mencionó al inicio del presente capítulo, todas las pruebas descritas en este documento se han repetido diez veces, para el caso de tiempos, y cinco veces, para el caso de bondad. En este caso particular, las cinco repeticiones han arrojado resultados idénticos, tanto para el caso estándar - aunque no se haya recogido aquí dicho análisis - como para el caso óptimo.

En consecuencia, se puede concluir que, efectivamente, el error de comunicación puede considerarse despreciable, puesto que no provoca ningún cambio sustancial en los mapas de abundancia obtenidos a la salida de la cadena de desmezclado espectral. Por todo ello, también puede concluirse que, desde el punto de vista de la bondad, la cadena implementada en un actor y la implementada en 5 actores pueden considerarse equivalentes.

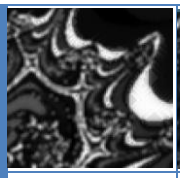
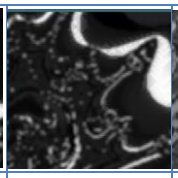
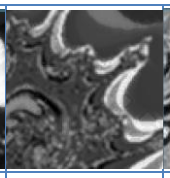
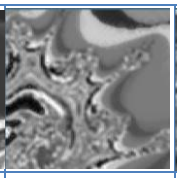
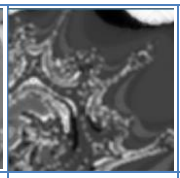
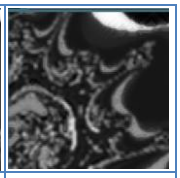
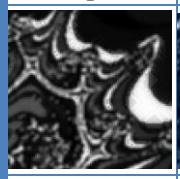
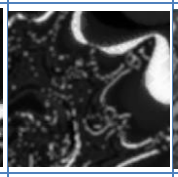
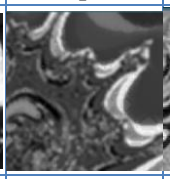
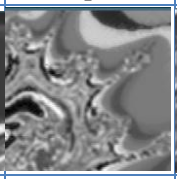
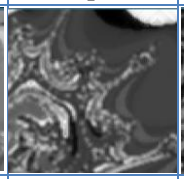
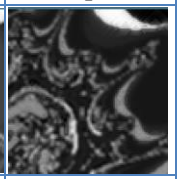
1 actor						
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5	Mapa 6
5 actores						
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5	Mapa 6

Tabla 4.9. Comparativa mapas de abundancia con y sin error de comunicación

Una vez comprobada la bondad de la cadena completa sin aleatoriedad, se ha de estudiar el efecto que ésta introduce en dicha cadena. Como ya se estudió en el apartado anterior, se había concluido que el efecto de la aleatoriedad ocasionaba, de forma genérica, un desorden en el conjunto de *endmembers* calculado y, en consecuencia, un desorden en los mapas de abundancia estimados. Por ello, en este apartado se tratará de probar que esta hipótesis se mantiene y que, por tanto, es acertada.

Al igual que en el caso anterior, las pruebas realizadas para comprobar esta suposición se han dividido en dos bloques: por un lado, se ha analizado un caso estándar, en el que la *fractal* 2 se ha reducido a 25 bandas y se han extraído 10 *endmembers* y, por el otro, se ha realizado el análisis de la misma imagen, pero para el caso óptimo, en el que tanto el número de bandas a las que se reducirá la imagen como el número de *endmembers* a calcular es 6.

Para facilitar la comprensión de este estudio, se va a realizar un análisis similar al desarrollado en [1] para explicar el efecto de la aleatoriedad en la configuración de un único actor. Para ello, primero se ha de explicar brevemente el concepto de la aleatoriedad, que se ha explicado detalladamente en [1]. A lo largo de este documento, se ha explicado que, en la fase de extracción de *endmembers*, se introduce una componente aleatoria que hace que los resultados obtenidos cada vez que se ejecuta la cadena varíen ligeramente. En un momento dado del algoritmo VCA – que es el algoritmo elegido en [1] para la implementación de la mencionada fase –, esta componente aleatoria genera unos índices que, en última instancia, influyen en la forma en que se rellena la matriz de salida de dicho algoritmo, que contiene el conjunto de *endmembers* extraídos. En [1] se ha comprobado que, en consecuencia, estos índices están directamente relacionados con los *endmembers* calculados, ya que cada uno de ellos representa de forma unívoca a cada *endmember* presente en la imagen.

Teniendo en cuenta el concepto de índice, el análisis de la aleatoriedad puede reducirse a recoger, para cada una de las pruebas realizadas, los índices generados por el algoritmo VCA y comprobar, tras ello, que en todas las pruebas aparecen los mismos índices, o que al menos el porcentaje de repetición es muy elevado, ya que eso sería equivalente a demostrar que en todas las pruebas aparecen los mismos *endmembers*, estén colocados o no.

Como se ha mencionado anteriormente, para comprobar la bondad de los distintos algoritmos y configuraciones se han repetido las pruebas un total de 5 veces. De todas ellas, se han extraído los índices utilizados por el algoritmo VCA, y se ha realizado una comparativa de los mismos. Para facilitar la comprensión de esta comparativa, los índices de cada prueba se han recogido en una tabla y se ha indicado, con una escala de colores, el grado de coincidencia de los distintos *endmembers* en cada una de las pruebas realizadas. Esta escala de colores se recoge en la tabla 4.10, donde puede observarse que, por ejemplo, el color rojo se corresponde con la repetición de un *endmember* en todas las pruebas realizadas, mientras que el color blanco indica que sólo está presente en una de las pruebas.

Repeticiones	Color
5	
4	
3	
2	
1	

Tabla 4.10. Escala de repeticiones para cinco pruebas

En la tabla 4.11 se recogen los índices extraídos para las cinco pruebas realizadas para el caso estándar; para cada prueba se han obtenido 10 índices, puesto que este caso calcula 10 *endmembers*. Como puede observarse, la gran mayoría de índices se repiten en todas las pruebas; en concreto, se puede comprobar que, en todas las pruebas, de los 10 índices coinciden 8, lo que supone un 80% de coincidencia entre los *endmembers* extraídos en las distintas pruebas. Además, se puede apreciar que, de los índices que no coinciden - 2 por prueba -, el 80% están presentes en 4 de las 5 pruebas realizadas.

Otro aspecto muy interesante a destacar es que, como puede observarse, las pruebas 1, 3 y 4 presentan exactamente los mismos índices, aunque en distinto orden. Como se acaba de mencionar, cada índice es representativo de un único *endmember*; en consecuencia, se puede afirmar que estas tres pruebas arrojan resultados idénticos en cuanto a mapas de abundancia. La única diferencia entre ellas será que estos índices están desordenados, lo que indica que, en consecuencia, sus mapas de abundancia también lo estarán - como se demostró en el apartado anterior-. Por último, respecto a las pruebas 2 y 5 también puede destacarse que, a pesar de no coincidir al 100%, ambas presentan un grado de coincidencia del 90% con las pruebas 1, 3 y 4, y un 80% entre ellas. En consecuencia, se puede concluir que, de forma global, el 60% de las pruebas arrojan resultados idénticos, aunque desordenados.

Orden de *endmember*	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1	3199	1613	1613	799	1613
2	4799	3199	799	4799	799
3	799	9403	9931	9963	3199
4	9963	799	4799	9403	9403
5	9403	9996	8679	3199	4799
6	1613	4799	9403	9931	8679
7	8679	9963	9963	8679	9931
8	9931	55	3199	1613	9963
9	9996	9931	9996	9996	7807
10	182	182	182	182	182

Tabla 4.11. Comparativa de endmembers *para el caso estándar de la cadena completa – 5 actores*

En la tabla 4.12 se presenta el mismo estudio, pero para el caso óptimo. Como puede deducirse, ahora el número de índices extraídos para cada prueba se ha reducido a 6, puesto que éste era el resultado arrojado por el algoritmo HYSIME - que proporcionaba el número óptimo de *endmembers* a calcular -. Realizando el mismo análisis que para el caso estándar, puede observarse que, para todas las pruebas realizadas, de los 6 índices calculados coinciden plenamente 5 de ellos, lo que supone un 83.33% de coincidencia entre las distintas pruebas realizadas. Además, en esta ocasión se puede apreciar que existen únicamente dos índices no coincidentes, y que uno de ellos - el 799 - se repite en 4 de las 5 pruebas realizadas; esto pone de manifiesto que, para este caso, las pruebas 1,2, 3 y 4 son idénticas, y que la 5 comparte con ellas 5 de los 6 *endmembers* calculados. En consecuencia, el 80% de las pruebas realizadas arrojan resultados equivalentes.

Como ya se ha mencionado anteriormente, la extracción de estos índices parece indicar que el desorden de los mapas de abundancia está directamente relacionado con el desorden que puede observarse en los índices extraídos para cada prueba. Para comprobar que esta suposición es correcta, se va a establecer la relación entre los índices de las pruebas 1 y 2 de la tabla 4.12; tras ello, se aportarán los mapas de abundancia de ambas pruebas para comprobar si la hipótesis es cierta y, en consecuencia, ésta es la razón por la que los mapas de abundancia aparecen en distinto orden cada vez que se ejecuta una prueba.

En concreto, los índices de estas pruebas se relacionan de la siguiente forma: en primer lugar, los índices situados en las posiciones 1, 3, 4 y 5 son coincidentes; respecto a las posiciones restantes, puede observarse que éstas están intercambiadas: la segunda posición de una coincide con la sexta de la otra, y viceversa.

Orden de *endmember*	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1	799	799	1613	9963	9403
2	9963	4799	3199	799	1613
3	3199	3199	799	3199	3199
4	1613	1613	4799	1613	4799
5	9931	9931	9963	9931	9963
6	4799	9963	9931	4799	9931

Tabla 4.12. Comparativa de endmembers *para el caso óptimo de la cadena completa – 5 actores*

A continuación, en la tabla 4.13 se proporcionan los mapas de abundancia extraídos de las pruebas 1 y 2 del caso óptimo. Efectivamente, se puede comprobar que las relaciones predichas en función de los índices extraídos son correctas, ya que los mapas 1, 3, 4 y 5 coinciden en ambas pruebas y, por su parte, el 2 y el 6 de ambas pruebas aparecen intercambiados.

Por tanto, gracias a estas pruebas se ha podido comprobar que, en efecto, puede establecerse que la influencia de la aleatoriedad produce, principalmente, un desorden en los mapas, si bien es cierto que, en algunas ocasiones, pueden llegar a producirse *endmembers* espurios que no se repiten en ninguna otra prueba.

Para finalizar el estudio de la bondad, se van a comparar los mapas de abundancia de estas dos pruebas con una de las pruebas realizadas en [1] para la configuración de un único actor. En el citado documento, el autor utiliza estas pruebas para comprobar la bondad de la cadena; en consecuencia, si dicha prueba coincide con las recogidas en la tabla 4.13, también se podrá confirmar la bondad de la cadena implementada con cinco actores.

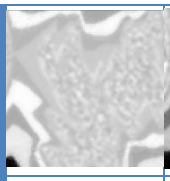
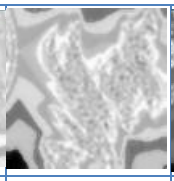
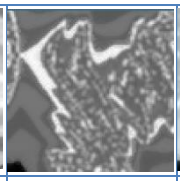
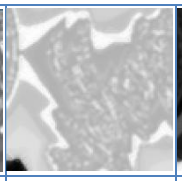
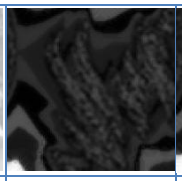
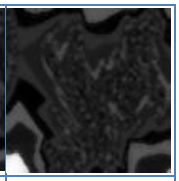
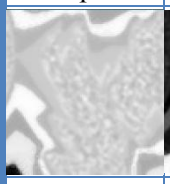
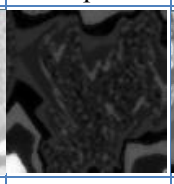
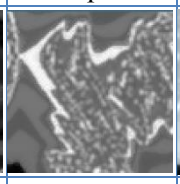
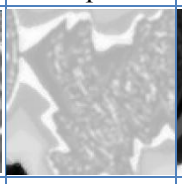
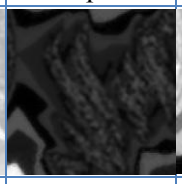
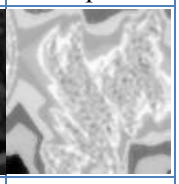
Prueba 1						
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5	Mapa 6
Prueba 2						
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5	Mapa 6

Tabla 4.13. Comparativa de mapas de abundancia para el caso óptimo de la cadena completa - 5 actores

En la tabla 4.14 se recogen tanto los mapas de abundancia para la prueba mencionada como los índices asociados a cada uno de ellos. Si se compara esta tabla con las dos anteriores, puede comprobarse que, efectivamente, la prueba para un actor coincide plenamente con las pruebas recogidas en la tabla 4.13, tanto en los mapas de abundancia como en los índices extraídos. En consecuencia, se puede afirmar que la bondad de la cadena está comprobada.

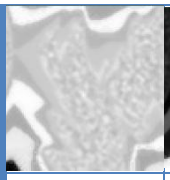
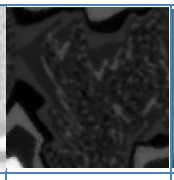
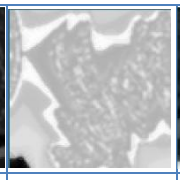
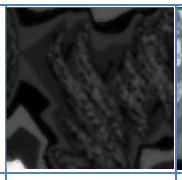
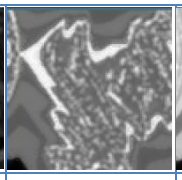
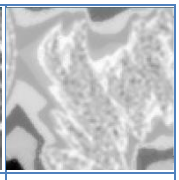
Prueba 1 actor						
	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Mapa 5	Mapa 6
Índice	799	4799	1613	9931	3199	9963

Tabla 4.14. Mapas de abundancia para una de las pruebas del caso óptimo - 1 actor

Para concluir este apartado, se va a realizar el análisis de los tiempos de ejecución de la cadena. Este análisis es especialmente importante, puesto que en él se va a estudiar cómo afecta la división de los procesadores a la velocidad de ejecución. En concreto, se van a estudiar dos casos distintos: por un lado, se compararán los tiempos obtenidos por Daniel Madroñal para la cadena completa en un único actor [1] con los tiempos extraídos en este documento para la cadena en cinco actores; tras ello, se aportarán las medidas de la velocidad de procesado para las distintas configuraciones de división de procesadores expuestas en el capítulo anterior.

En la tabla 4.15 se recogen los resultados obtenidos para el primer caso, en el que se ha medido el tiempo de procesado de la *fractal* 2 en tres situaciones distintas: los casos que menor y mayor carga computacional conllevan, respectivamente, y el caso óptimo, anteriormente explicado. La conclusión principal que puede extraerse de esta tabla es que la comunicación entre actores no introduce un retardo significativo, ya que la variación de tiempos entre una configuración y otra es de apenas unas centésimas de segundo; de hecho, en algunos casos es incluso más rápida la configuración de cinco actores que la de un único actor. De forma genérica, el impacto de la comunicación en el tiempo de ejecución aumenta según lo hace la carga computacional - o, lo que es lo mismo, el volumen de datos a analizar -. En la última columna, se calcula la diferencia de los tiempos medios entre ambas configuraciones; como puede observarse, a medida que aumenta esta complejidad, la diferencia entre una y otra va creciendo, hasta tal punto que, en el caso más lento, la cadena con 5 actores tarda 16 milisegundos más que la de un actor, mientras que en el resto de casos el resultado era inverso.

Caso	Características		1 actor	5 actores	Diferencia
Caso más rápido	25 bandas 5 *endmembers*	Mínimo	10.571357	10.557273	-0.0150041
		Media	10.5899674	10.5749633	
		Máximo	10.617797	10.58281	
Caso más lento	100 bandas 20 *endmembers*	Mínimo	12.231839	12.244452	0.0161762
		Media	12.2554437	12.2716199	
		Máximo	12.285597	12.283978	
Caso óptimo	6 bandas 6 *endmembers*	Mínimo	10.509184	10.488279	-0.013902
		Media	10.5136651	10.4997631	
		Máximo	10.525474	10.505216	

Tabla 4.15. Comparativa de tiempos (en segundos) de las dos configuraciones

Por último, se va a realizar el estudio de los tiempos de ejecución de la cadena de desmezclado espectral para las 16 posibles divisiones en procesadores explicadas en el capítulo anterior. Para facilitar la comprensión de este análisis, dichas posibilidades se repiten en la tabla 4.16.

Actores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Source	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
PCA	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0
VCA	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0
LSU	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1
Display	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1
0 - procesador 1									*1 - procesador 2*							

Tabla 4.16. División en procesadores

Utilizando la misma numeración, en la tabla 4.17 se recoge el análisis realizado. Para cada una de las divisiones, se ha medido el tiempo de ejecución de la cadena completa un total de 10 veces, incluyendo las interfaces de entrada y salida. Respecto a la información aportada en la tabla, se ha recogido, por un lado, la media de las 10 ejecuciones para cada una de las divisiones y, por otro, la diferencia con la división 1, que se corresponde con una ejecución mononúcleo - todos los actores en un mismo procesador.

El hecho que más llama la atención de la información recogida en la tabla 4.17 es que, de forma contraria a las suposiciones realizadas, el caso más rápido es, precisamente, el caso en el que todos los actores se ejecutan en un mismo procesador. El problema que parece ocasionar este resultado es que la división de actores diseñada no es óptima. Esto se debe a que, como se ha podido apreciar a lo largo de este documento, los actores implementados se ejecutan, principalmente, de forma secuencial, sin que uno pueda empezar hasta que haya terminado el otro. En consecuencia, con esta división de actores la paralelización no es eficiente y, por tanto, las distintas pruebas realizadas demuestran que la velocidad de procesado no se incrementa, sino que disminuye. Conviene destacar que, al ser una primera versión, esta elección de actores no se realizó buscando el reparto eficiente respecto a la paralelización, sino que se buscaba facilitar la modificación o sustitución de los distintos algoritmos sin afectar al resto de etapas.

La solución a este problema sería, por tanto, estudiar otras posibles distribuciones de actores. A este respecto, cabe destacar que el camino a seguir sería la atomización de cada algoritmo en distintos actores; puesto que los algoritmos se han de ejecutar de forma secuencial, la forma efectiva de paralelizar el sistema sería tratar de localizar aquellas tareas que puedan ejecutarse

de forma simultánea en un algoritmo y, tras ello, englobarlas en distintos actores. Así, podría asignarse cada una de estas tareas a un procesador distinto y, en consecuencia, podrían ejecutarse en paralelo, reduciendo así el tiempo global del algoritmo. Conviene resaltar que, a este respecto, los algoritmos que necesitan en mayor medida esta paralelización son HYSIME y PCA, puesto que, como ha podido comprobarse en los tiempos extraídos tanto en este Proyecto como en [1], son los que más tiempo necesitan para ejecutarse.

División	Media	Diferencia	División	Media	Diferencia
1	13.5456954	0	***9***	13.9219444	0.376249
2	13.943479	0.3977836	***10***	14.1114418	0.5657464
3	14.0032496	0.4575542	***11***	13.952134	0.4064386
4	13.6958592	0.1501638	***12***	13.8819096	0.3362142
5	13.874145	0.3284496	***13***	14.1128882	0.5671928
6	13.728713	0.1830176	***14***	13.753625	0.2079296
7	13.8043064	0.258611	***15***	13.7499592	0.2042638
8	13.9315906	0.3858952	***16***	13.8346968	0.2890014

Tabla 4.17. Tiempos de ejecución (en segundos) de las distintas divisiones en procesadores

No obstante, a pesar de no haber mostrado una mejora temporal con la división en procesadores, se puede considerar que los objetivos planteados se han cumplido; el objetivo de este análisis era demostrar que, efectivamente, RVC - CAL permite realizar distribuciones de hilos en distintos procesadores de forma rápida y sencilla y, que esto puede ser aprovechado para diseñar, de forma efectiva, la paralelización del sistema. Considerando que, gracias a este análisis, se ha podido descartar rápidamente la distribución etapa - actor, se puede concluir que ésta es una buena herramienta para la búsqueda de una distribución eficiente.

Además, cabe destacar que la utilización de RVC - CAL ha permitido la reducción del tiempo de ejecución de todos los algoritmos de la cadena - en comparación con los tiempos del software *Hypermix* -. Esto, unido al hecho de que se puede diseñar de forma eficiente la paralelización del sistema, parece arrojar resultados optimistas al respecto del objetivo de la consecución del tiempo real. Como se verá en el próximo capítulo, este diseño de la paralelización eficiente constituirá una de las líneas de trabajo futuras.

4.3. Análisis de rendimiento

Para concluir este capítulo, se va a proceder a realizar un estudio del rendimiento de la cadena, analizando los recursos consumidos en cada actor. En concreto, este estudio se realizará para el caso óptimo, en el que, para la *fractal* 2, se extraen 6 *endmembers*. Dicho análisis se ha dividido en dos partes diferenciadas:

- En primer lugar, se realiza un análisis básico del rendimiento de la cadena de desmezclado espectral en sus dos configuraciones posibles - un actor y 5 actores -, pero para una ejecución en un único procesador. Este análisis será el desarrollado en el presente Proyecto Fin de Grado.
- En segundo lugar, se realiza un estudio detallado del consumo de recursos de la cadena completa ejecutada en varios procesadores. Para ello, se decidió escoger los casos mejor y peor en cuanto a velocidad de ejecución de las distintas distribuciones multinúcleo analizadas en el caso anterior. Este análisis se ha desarrollado en el Proyecto Fin de Grado de Daniel Madroñal [1].

Cabe destacar que, para realizar ambos análisis, se ha utilizado la herramienta *papify*, ya mencionada anteriormente, que aplica la librería PAPI al sistema bajo estudio para medir el consumo de recursos del mismo. Asimismo, también se ha hecho uso de otra herramienta - denominada *papiplot*[21] y también desarrollada por Alejo Iván Arias - que facilita la interpretación de los resultados obtenidos por *papify*, proporcionando los datos en tablas y gráficas de barras.

En consecuencia, en este apartado se realizará un análisis sencillo del consumo de recursos de la cadena completa, cuando ésta se ejecuta únicamente en un procesador; asimismo, se realizará una comparativa entre los recursos consumidos por la configuración en un actor y por la configuración en 5 actores, al igual que se hizo con el análisis temporal. Para hacer esta comparativa, se han monitorizado los registros de PAPI recogidos en la tabla 4.18.

Registro	Definición
PAPI_TOT_INS	Instrucciones ejecutadas en el procesador
PAPI_TOT_CYC	Ciclos de reloj empleados en la ejecución
PAPI_BR_INS	Instrucciones totales condicionantes (*if, else*) ejecutadas
PAPI_FP_INS	Instrucciones totales de punto flotante ejecutadas
PAPI_FML_INS	Instrucciones totales de multiplicación de datos en punto flotante
PAPI_VEC_INS	Instrucciones vectoriales totales ejecutadas
PAPI_FDV_INS	Instrucciones totales de división de datos en punto flotante
PAPI_L1_DCM	Accesos fallidos a memoria caché de datos
PAPI_L1_ICM	Accesos fallidos a memoria caché de instrucciones
PAPI_L1_TCM	Accesos fallidos totales a memoria caché

Tabla 4.18. Registros PAPI monitorizados

Estos registros han sido monitorizados para las dos configuraciones mononúcleo utilizadas durante este Proyecto Fin de Grado - 1 actor y 5 actores -. De esta forma, se puede proporcionar una comparativa entre los dos, puesto que la división de actores realizada en la segunda configuración es equivalente a la división en acciones de la primera - esto es, la configuración en un único actor contiene acciones equiparables a los actores *PCA*, *VCA* y *LSU* -. Por ello, la comparativa se va a realizar siguiendo la estructura de algoritmos: en primer lugar, se comparará el actor *PCA* de la configuración en 5 actores con la acción homónima de la configuración de un solo actor; tras ello, se realizará lo propio con VCA y, por último, con LSU. Cabe destacar que se han omitido de este estudio las acciones y actores relacionados con interfaz de entrada y salida porque el objetivo es comparar el consumo de recursos de los algoritmos en sí. Por esta misma razón, para el caso de 5 actores no se tienen en cuenta las acciones relativas a la comunicación, sino únicamente las relacionadas con el algoritmo en sí.

En la tabla 4.19 se recogen las comparativas de los algoritmos PCA, VCA y LSU. Si se comparan estas tablas, la conclusión general que puede extraerse es que las mediciones realizadas son muy similares para las dos configuraciones estudiadas. Este resultado es coherente puesto que, como se acaba de explicar, lo que se está comparando es el algoritmo en sí. Por ello, lo que estas tablas comprueban es que, en general, el consumo del algoritmo se puede considerar constante, independientemente de la configuración implementada.

Además, otro aspecto relevante que puede extraerse de estas tablas es que, en general, el consumo de recursos del algoritmo PCA es exponencialmente mayor que el consumo asociado al resto de los algoritmos; por el contrario, el algoritmo que menos recursos consume es el LSU.

[21] https://github.com/alejoar/papiplot

Estos resultados son coherentes con el análisis temporal realizado tanto en este documento como en [1], ya que, según el mismo, el algoritmo que más tiempo requería para ejecutarse era el PCA, mientras que, por el contrario, el que menos tiempo necesitaba era el LSU.

Registro	PCA		VCA		LSU	
	1 actor	5 actores	1 actor	5 actores	1 actor	5 actores
PAPI_TOT_INS	28093897349	28093897526	57880239	57817580	11480306	11409615
PAPI_TOT_CYC	29149949315	29873373862	38337377	38329791	8147562	7622669
PAPI_L1_DCM	656534462	656651346	613905	613728	48672	46652
PAPI_L1_ICM	9396	25933	2942	3091	2490	576
PAPI_L1_TCM	656539720	656571713	619707	620312	54319	48238
PAPI_BR_INS	1031946369	1031936052	3707217	3707064	570685	555265
PAPI_FP_INS	6093068120	6093237099	10545277	10551184	2478458	2482868
PAPI_FML_INS	3058189596	3058287150	4625837	4569062	1148683	1150996
PAPI_VEC_INS	990597473	990597921	2399088	2398891	482616	482496
PAPI_FDV_INS	599492	599486	1657	1599	521	446

Tabla 4.19. Comparativa total sin comunicación

Para observar el consumo de recursos asociado a la comunicación, en la tabla 4.20 se recoge el mismo análisis que el realizado para cada etapa, pero considerando también los recursos consumidos por las acciones de comunicación. Como puede observarse, ahora la diferencia entre las dos configuraciones es sustancial; para todos los registros medidos, la configuración de 5 actores consume más recursos que la de un actor.

En concreto, se puede resaltar el registro *PAPI_L1_ICM*, cuya diferencia entre configuraciones ha aumentado varios órdenes de magnitud. Esto puede explicarse teniendo en consideración el aumento de la complejidad de la red o *network*; esto hace que el volumen de instrucciones crezca considerablemente - como puede apreciarse en el registro *PAPI_TOT_INS*, que es el encargado de medir el total de instrucciones - y que, en consecuencia, la memoria caché dedicada a las instrucciones se llene con más frecuencia, traduciéndose en un aumento de este registro. Por el contrario, se puede observar que el registro asociado a la memoria caché de datos - *PAPI_L1_DCM* -, aunque ha aumentado, se mantiene en el mismo orden de magnitud. Esto se debe a que el volumen de datos que necesita la red - constantes, variables, etc - sigue manteniéndose más o menos constante pese al aumento de la complejidad.

Registro	PCA		VCA		LSU	
	1 actor	5 actores	1 actor	5 actores	1 actor	5 actores
PAPI_TOT_INS	28093897349	28206016403	57880239	63622275	11480306	23614620
PAPI_TOT_CYC	29149949315	29978678759	38337377	53054274	8147562	40515561
PAPI_L1_DCM	656534462	657693177	613905	869940	48672	686394
PAPI_L1_ICM	9396	632389	2942	268869	2490	591916
PAPI_L1_TCM	656539720	658908133	619707	1485667	54319	1907810
PAPI_BR_INS	1031946369	1043153198	3707217	4929093	570685	3088188
PAPI_FP_INS	6093068120	6095524878	10545277	10551220	2478458	2873244
PAPI_FML_INS	3058189596	3060514428	4625837	4569062	1148683	1271396
PAPI_VEC_INS	990597473	992867921	2399088	2458927	482616	672532
PAPI_FDV_INS	599492	599486	1657	1599	521	446

Tabla 4.20. Comparativa total con comunicación

Por último, para concluir este capítulo se va a proporcionar una comparativa del consumo de recursos de la configuración de 5 actores. De esta forma, se podrá observar la proporción del consumo asociado a los algoritmos en sí y la asociada a las interfaces de entrada y salida. Además, como la magnitud de los resultados obtenidos dificulta la extracción de conclusiones, esta comparativa se proporcionará en un diagrama de barras.

Así, en la figura 4.1 se puede observar los resultados extraídos para algunos de los registros monitorizados. El motivo por el que no se han expuesto todos los registros es que el carácter de esta comparativa es puramente ilustrativo, por lo que sólo se han expuesto algunos de los registros más significativos - como, por ejemplo, *PAPI_TOT_INS* y *PAPI_TOT_CYC*, que aportan información sobre el total de instrucciones y de ciclos de reloj requeridos.

Teniendo en consideración que el eje vertical de la figura es logarítmico, en la figura 4.1 puede observarse que, efectivamente, la diferencia entre los recursos consumidos por el algoritmo PCA y los consumidos por el resto es exponencial. Además, otro aspecto significativo es que, como puede observarse, los actores *Source* y *Display* - correspondientes a las interfaces de entrada y salida, respectivamente - también consumen una considerable cantidad de recursos. Esto se debe a que la carga computacional de estos actores es bastante elevada, ya que tienen que leer la imagen hiperespectral -que, como se mencionó en capítulos anteriores, contiene un volumen de datos muy elevado - y escribir los mapas de abundancias.

En conclusión, resulta evidente que el algoritmo PCA constituye un cuello de botella en la cadena de procesado de imágenes hiperespectrales, ya que, como puede comprobarse tanto en esta figura como en las tablas anteriores, el consumo de recursos asociado al PCA es exponencialmente superior al del resto de algoritmos. Por tanto, se puede afirmar que este algoritmo ha de ser paralelizado u optimizado para poder alcanzar el objetivo de tiempo real. Además, respecto a la cadena completa - y no únicamente a la algoritmia -, se observa que otro posible cuello de botella se encuentra en la interfaz de entrada y, en menor medida, en la de salida, por lo que también son susceptibles de ser optimizadas.

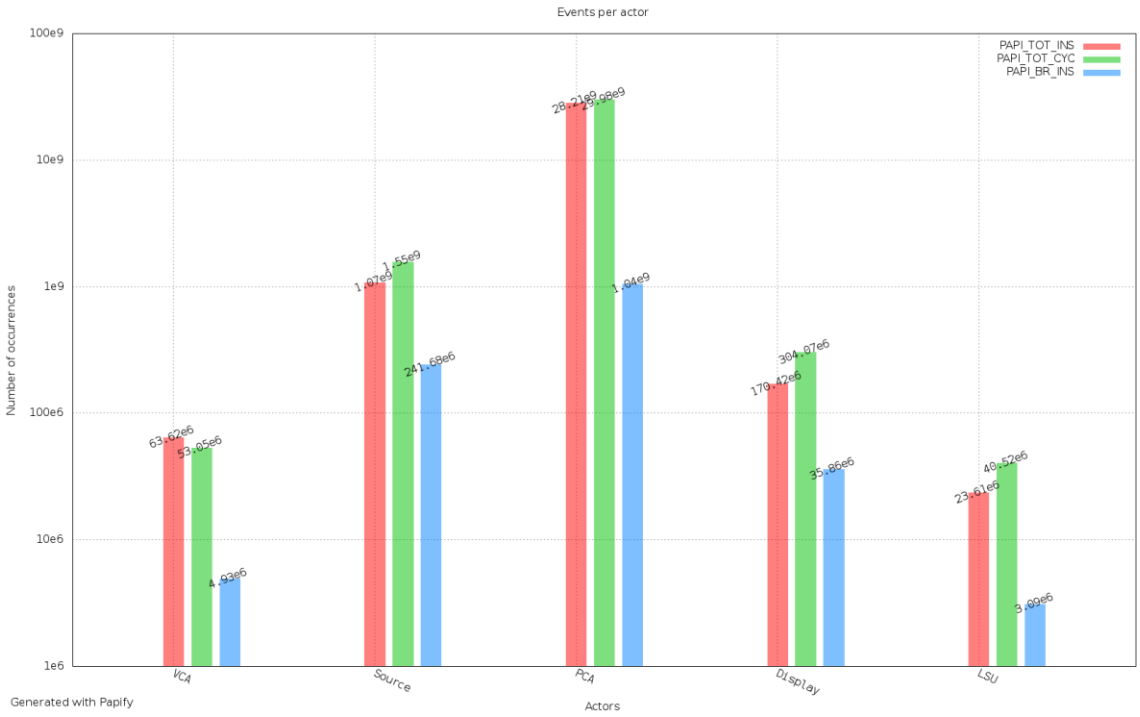


Fig. 4.1. Comparativa de recursos consumidos por cada actor en la configuración de 5 actores

CAPÍTULO 5. CONCLUSIONES

5.1.Conclusiones

Como consecuencia del análisis de los resultados obtenidos para la utilización de la librería desarrollada a lo largo de este Proyecto Fin de Grado, se han podido extraer una serie de conclusiones específicas, que se exponen a continuación.

- En primer lugar, respecto a la bondad de la librería implementada, en el análisis por etapas se ha comprobado que, efectivamente, los resultados obtenidos gracias a la utilización de la librería coinciden completamente con los proporcionados por la herramienta software de procesado de imágenes hiperespectrales *Hypermix*, por lo que queda comprobado el correcto funcionamiento de la misma. Conviene destacar que, de forma genérica, la principal diferencia detectada en las ejecuciones de los algoritmos ha sido un desorden de los mapas de abundancia con respecto a los generados por *Hypermix*. Este desorden surge como consecuencia de la componente aleatoria asociada al algoritmo VCA de la fase de extracción de *endmembers* y no ocasiona ningún efecto indeseado en la cadena de desmezclado espectral.

- Por su parte, este análisis también ha permitido comprobar que el tiempo de ejecución de los algoritmos implementados se ha reducido considerablemente, llegando a conseguir aceleraciones cercanas al 30%, en el caso de la fase de estimación de *endmembers*, y al 60%, en el caso de la etapa de estimación de abundancias. Cabe destacar que esta reducción temporal se debe exclusivamente a la utilización de la librería, puesto que estos resultados se han obtenido en una ejecución mononúcleo.

- El tiempo mínimo obtenido en la ejecución de la cadena completa se ha situado en los 10 segundos, aproximadamente - se ha de resaltar que, de este tiempo, un gran porcentaje se debe exclusivamente al algoritmo PCA -. Respecto al objetivo de tiempo real del proyecto HELICoiD, se puede concluir que éste se cumple si el sistema de adquisición de la cámara necesita, como mínimo, ese tiempo para capturar una imagen. Este aspecto no ha podido evaluarse porque el tiempo de adquisición en el entorno de HELICoiD todavía se desconoce, por lo que esta problemática se tratará en profundidad en futuras líneas de trabajo.

- El estudio de la ejecución de la cadena en varios núcleos ha permitido demostrar el potencial de RVC - CAL para realizar distribuciones de hilos en distintos procesadores de forma rápida y sencilla, proporcionando así una buena herramienta para la búsqueda de un diseño eficiente de la paralelización del sistema. Asimismo, dicho estudio ha permitido también comprobar que los algoritmos implementados están diseñados para una ejecución secuencial, por lo que la posibilidad de ejecutar cada algoritmo en un núcleo no aporta paralelismo a la cadena. Por tanto, este paralelismo ha de buscarse de forma interna en cada algoritmo; en concreto, el algoritmo más susceptible de ser paralelizado es PCA, puesto que es el que más tiempo y recursos necesita.

- Por último, el análisis del consumo de recursos realizado con la herramienta *papify* arroja resultados concordantes con los tiempos de ejecución obtenidos. Asimismo, comprueba la hipótesis de que el algoritmo PCA en sí representa un cuello de botella y que, en consecuencia, debe ser optimizado para reducir el tiempo asociado al mismo. Este estudio también resalta la existencia de cuellos de botella en las interfaces de entrada y salida, debido al gran volumen de datos que deben leer o escribir.

Como conclusión general, se puede resaltar que, gracias a la aplicación de la librería desarrollada, se ha conseguido construir una cadena completa en RVC - CAL que, además de proporcionar los mismos resultados que otras herramientas - como *Hypermix* -, también presenta una mejora sustancial en el tiempo de ejecución. Asimismo, esta librería consigue reducir sustancialmente tanto la complejidad como la dedicación requerida a la hora de implementar los algoritmos, por lo que se puede concluir que, efectivamente, la librería construida es una herramienta con un gran potencial para realizar un análisis hiperespectral rápido y sencillo - y, por tanto, eficiente.

5.2. Líneas futuras

La finalización de este Proyecto Fin de Grado y su posterior evaluación ha puesto de manifiesto la existencia de algunas líneas de trabajo futuras que han quedado abiertas, y que van a ser explicadas a continuación. Conviene mencionar que esta evaluación ha sido realizada de forma conjunta con Daniel Madroñal, por lo que las líneas planteadas pueden considerarse comunes a ambos proyectos.

- Estudiar nuevos algoritmos de procesado de imágenes hiperespectrales, con el fin de ampliar la librería y proporcionar nuevas funcionalidades. En concreto, debido a la naturaleza de HELICoiD, existe un gran interés en estudiar nuevos algoritmos que permitan el análisis de la imagen hiperespectral a medida que el sensor va capturándola; de este modo, no habría que esperar a tener la imagen completa para empezar a analizarla, reduciendo así el tiempo necesario para obtener resultados.

- Aprovechar las funcionalidades de RVC - CAL a la hora de realizar distribuciones de hilos en distintos procesadores para tratar de encontrar la paralelización óptima de los algoritmos más problemáticos en cuanto al tiempo de ejecución. De esta forma, podrá plantearse otra red o *network*, más eficiente desde el punto de vista de la paralelización. Como la red diseñada en este Proyecto ofrecía ventajas relevantes relacionadas con la facilidad de modificación o sustitución de algoritmos, otra posible línea de actuación es desarrollar algoritmos que puedan paralelizarse de forma simultánea, evitando así la modificación de la red.

- Buscar una alternativa que permita eliminar las librerías utilizadas en este Proyecto para el procesado matricial - ITK, VNL y OTB, mencionadas en el capítulo 3 de este documento y explicadas en detalle en [1] -. Esto permitiría una reducción considerable en la complejidad de implementación de la librería RVC - CAL, puesto que ésta podría codificarse únicamente en C, evitando así la necesidad de compilar simultáneamente C y C++. Con esta mejora también se conseguiría aumentar la interoperabilidad de la librería para las plataformas existentes en la actualidad.

- Implementar la cadena de desmezclado espectral - junto a la librería desarrollada - en la plataforma multinúcleo que se seleccione para el proyecto HELICoiD; de esta forma, se podrá aportar una idea más concisa de las posibilidades de paralelización del sistema, ya que los resultados preliminares recogidos en este Proyecto han sido obtenidos en un ordenador sencillo - procesador *Intel Core Dúo*.

- Tras la finalización de la línea anterior, se procederá a estudiar el consumo de energía de la cadena de desmezclado en diversas plataformas multinúcleo, con el fin de poder desarrollar analizadores hiperespectrales portátiles.

REFERENCIAS

[1] D. Madroñal, "Generación de una librería RVC – CAL para la etapa de determinación de *endmembers* en el proceso de análisis de imágenes hiperespectrales", Proyecto Fin de Grado, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, Universidad Politécnica de Madrid, 2014.

[2] A. F. H. Goetz, G. Vane, J. E. Solomon, B. N. Rock, "Imaging spectrometry for Earth remote sensing", *Science*, 228, pp. 1147–1153, 1985.

[3] D. Manolakis, G. Shaw, "Detection algorithms for hyperspectral imaging applications", *IEEE Signal Processing Magazine*, vol. 19, 2002.

[4] D. Landgrebe, "Hyperspectral Image Data Analysis", *IEEE Signal Processing Magazine*, vol. 19, no. 1, 2002.

[5] J. Bioucas-Dias , A. Plaza , G. Camps-Valls , P. Scheunders , N. Nasrabadi, J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges", *IEEE Geosci. Remote Sens. Mag*., vol. 1, no. 2, pp. 6 -36, 2013

[6] R.O. Green, et al., "Imaging Spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)", *Remote Sensing of Environment*, vol. 65, pp. 227-248, 1998.

[7] A. Plaza, Q. Du, J. Bioucas-Dias, X. Jia, F. Kruse, "Foreword to the special issue on spectral unmixing of remotely sensed data", *IEEE Trans. Geosci. and Remote Sens.*, vol. 49, no. 11, pp. 4103–4110, 2011.

[8] N. Keshava, J.F. Mustard, "Spectral unmixing", *IEEE Signal Processing Magazine*, vol. 19, pp. 44-57, 2002.

[9] A. Plaza, P. Martinez, R. Perez, J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 3, pp. 650–663, 2004.

[10] N. Keshava, "A survey of spectral unmixing algorithms," *Lincoln Lab. J*., vol. 14, no. 1, pp. 55–78, 2003.

[11] S. Sánchez, " Diseño e implementación de una cadena completa para desmezclado de imágenes hiperespectrales en tarjetas gráficas programables (GPUs)", tesis doctoral, Dept. de Tecnología de los Computadores y de las Comunicaciones, Escuela Politécnica de Cáceres, Universidad de Extremadura, 2013.

[12] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5, pp. 354–379, 2012.

[13] J. M. Bioucas-Dias, J. M. P. Nascimento, "Hyperspectral subspace identification", *IEEE Trans. Geosci. Remote Sensing*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.

[14] L.I. Jiménez, "*Hypermix*: Una nueva herramienta libre para el desmezclado de imágenes hiperespectrales de la superficie terrestre", Trabajo Fin de Máster, Escuela Politécnica de Cáceres, Universidad de Extremadura, 2012.

[15] C. Chang, Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery", *IEEE Trans. Geosci. and Remote Sens.*, vol. 42, no. 3, pp. 608–619, 2004.

[16] A. Plaza, P. Martinez, R. Perez, J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations", *IEEE Trans. Geosci. and Remote Sens.*, 40, pp.2025–2041, 2002.

[17] M. D. Iordache, J. Bioucas-Dias, A. Plaza, "Sparse unmixing of hyperspectral data", *IEEE Trans. Geosci. and Remote Sens.*, vol. 49, no. 6, pp. 2014–2039, 2011.

[18] D. Ragona, B. Minster, T. Rockwell, J. Jussila, "Field imaging spectroscopy: A new methodology to assist the description, interpretation, and archiving of paleoseismological information from faulted exposures", *J. Geophys. Res.*, 111, B10309, 2006.

[19] F.A. Kruse, J.W. Boardman, J.F. Huntington, "Comparison of airborne hyperspectral data and EO-1 Hyperion for mineral mapping", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 6, pp. 1388–1400, 2003.

[20] J.P. Bibring, et al., "Global mineralogical and aqueous Mars history derived from OMEGA/Mars Express data", *Science*, 312, pp. 400–404, 2006.

[21] S.M. Pelkey et al., "CRISM multispectral summary products: Parameterizing mineral diversity on Mars from reflectance", *J. Geophys. Res*., vol. 112, 2007.

[22] S. Murchie et al, "Compact Reconnaissance Imaging Spectrometer for Mars (CRISM) on Mars Reconnaissance Orbiter (MRO)", *J. Geophys. Res.*, 112, E05S03, 2007.

[23] F. Poulet, J.-P. Bibring, J. F. Mustard, A. Gendrin, N. Mangold, Y. Langevin, R. E. Arvidson, B. Gondet, C. Gomez, and the OMEGA Team, "Phyllosilicates on Mars and implications for the early Mars history", *Nature*, 438, pp. 570 – 571, 2005.

[24] M.T. Eismann et al., "Target detection in desert backgrounds: infrared hyperspectral measurements and analysis", *Proc. SPIE 2561, Signal and Data Processing of Small Targets 1995*, vol. 80, 1995.

[25] M.T. Eismann et al., "Comparison of infrared imaging hyperspectral sensors for military target detection applications", *Proc. SPIE 2819, Imaging Spectrometry II,* vol. 91, 1996.

[26] D. Manolakis, S. Golowich, R. DiPietro, "Long-Wave Infrared Hyperspectral Remote Sensing of Chemical Clouds", *IEEE Signal Processing Magazine*, July 2014.

[27] J. Xing, C. Bravo, P. Jancsók, H. Ramon, J. De Baerdemaeker, "Detecting bruises on 'Golden Delicious' apples using hyperspectral imaging with multiple wavebands", *Biosystems Engineering*, vol. 90, no. 1, pp. 27-36, 2005.

[28] A.A. Gowen, C.P. O'Donnell, P.J. Cullen, G. Downey, J.M. Frías, "Hyperspectral Imaging - an emerging process analytical tool for food quality and safety control", *Trends in Food Science & Technology*, vol. 18, no. 12, pp. 590–598, 2007.

[29] J. Gómez-Sanchis, et al., "Hyperspectral computer vision system for the detection of Penicillium digitatum in citrus packing lines", *2004 CIGR International Conference, Beijing, China*, pp. 11 -14, 2004.

[30] J. Gómez-Sanchis et al., "Hyperspectral system for early detection of rottenness caused by Penicillium digitatum in mandarins", *Journal of Food Engineering*, vol. 89, no. 1, pp. 80–86, 2008.

[31] G. Berben, et al., "Methods of detection, species identification and quantification of processed animal proteins in feedingstuffs", *Base*, 13, pp. 59–70, 2009.

[32] C. Nansen, T. Herrman, R. Swanson, "Machine vision detection of bonemeal in animal feed samples", *Applied Spectroscopy,* vol. 64, no. 6, pp. 637–643, 2010.

[33] D.M. Tralli, R.G. Blom, V. Zlotnicki, A. Donnellan, D.L. Evans, “Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards”, *ISPRS J Photogramm Remote Sens* vol. 59, no. 4, pp. 185–198, 2005.

[34] M.F. Fingas, C.E. Brown, "Review of oil spill remote sensing,” *in Proceedings of the Fifth International Conference on Remote Sensing for Marine and Coastal Environments*, Environmental Research Institute of Michigan, Ann Arbor, Michigan, pp. 211-218, 2000.

[35] F. Salem, M. Kafatos, T. El-Ghazawi, R. Gomez, R. Yang, “Hyperspectral image analysis for oil spill detection,” *in Summaries of NASA/JPL Airborne Earth Science Workshop*, Pasadena, CA, 2002.

[36] M.N. Jha, J. Levy, Y. Gao, “Advances in remote sensing for oil spill disaster management: state-of-the-art sensors technology for oil spill surveillance”, *Sensors*, vol. 8, no. 1, pp. 236–255, 2008.

[37] J. Ellis, H.H. Davis, J.A. Zamudio, “Exploring for onshore oil seeps with hyperspectral imaging”, *Oil Gas J.*, vol. 99, no. 37, pp. 49–55, 2001.

[38] J. Plaza, R. Pérez, A. Plaza, P. Martínez, D. Valencia, “Mapping oil spills on sea water using spectral mixture analysis of hyperspectral image data”, *Proc. of SPIE 2001, 5995, 599509–1*, 2001.

[39] M. Lennon, V. Mariette, A. Coat, V. Verbeque, P. Mouge, G.A. Borstad, P. Willis, R. Kerr, M. Alvarez, “Detection and mapping of the november 2002 Prestige tanker oil spill in Galicia, Spain, with the airborne multispectral CASI sensor,” *in Proc. Third EARSeL Imaging Spectroscopy Workshop*, Herrsching, Germany, 2003.

[40] K. Clifford, A. Robinson, D. Miller, M. Davis, “Overview of sensors and needs for environmental monitoring”, *Sensors*, vol. 5, no. 2, pp. 4-37, 2005.

[41] D. J. Williams et al. "Detection and identification of toxic air pollutants using airborne LWIR hyperspectral imaging", *Proc. SPIE 5655, Multispectral and Hyperspectral Remote Sensing Instruments and Applications II*, 134, January 2005.

[42] CJ Keith, KS Repasky, RL Lawrence, SC Jay, JL Carlsten, “Monitoring effects of a controlled subsurface carbon dioxide release on vegetation using a hyperspectral imager”, *International Journal of Greenhouse Gas Control*, vol. 3, no. 5, pp. 626-632, 2009.

[43] M. Zhang, Z. Qin, X. Liu, S.L. Ustin, "Detection of stress in tomatoes induced by late blight disease in California, USA, using hyperspectral remote sensing", *International Journal of Applied Earth Observation and Geoinformation*, vol. 4, no. 4, pp. 295-310, 2003.

[44] H.H. Muhammed, "Hyperspectral crop reflectance data for characterising and estimating fungal disease severity in wheat", *Biosystems Engineering*, vol. 91, no. 1, pp. 9-20, 2005.

[45] V.E. Brando, G.A. Dekker, "Satellite hyperspectral remote sensing for estimating estuarine and coastal water quality", *Geoscience and Remote Sensing, IEEE Transactions,* vol. 41, no. 6, pp. 1378-1387, 2003.

[46] G. Edelman, E. Gaston, T. van Leeuwen, P. Cullen, M. Aalders, "Hyperspectral imaging for non-contact analysis of forensic traces," *Forensic Science International*, vol. 223, pp. 28-39, 2012.

[47] C. Fischer, I. Kakoulli, “Multispectral and hyperspectral imaging technologies in conservation: current research and potential applications”, *Reviews in Conservation*, 7, pp. 3–16, 2006.

[48] K. Knox, C. Dickinson, L. Wei, R. Easton, R. Johnston, "Multispectral imaging of the Archimedes Palimpsest", *in Proceedings of PICS 2001: Image Processing, Image Quality, Image Capture Systems Conference, Montreal,Quebec, Canada, Society for Imaging Science and Technology, Springfield*, pp. 206–210, 2001.

[49] R. Easton, W. Christens-Barry, K. Knox, "Spectral Image Processing and Analysis of the Archimedes Palimpsest", *19th European Signal Processing Conference, EUSIPCO* 2011.

[50] R.L. Easton Jr., W.G. Noel, "Infinite Possibilities: Ten years of study of the Archimedes Palimpsest", *Proc. Am. Philosophical Soc.*, vol. 154, pp. 50-76, 2010.

[51] R. Schultz, T. Nielsen, J. Zavaleta, R. Ruch, R. Wyatt, H. Garner, "Hyperspectral imaging: A novel approach for microscopic analysis", *Cytometry*, vol. 43, pp. 239 – 247, 2001.

[52] Y. Roggo, A. Edmond, P. Chalus, M. Ulmschneider, "Infrared hyperspectral imaging for qualitative analysis of pharmaceutical solid forms", *Analytica Chimica Acta*, vol. 535, no. 1, pp. 79-87, 2005.

[53] G. Lu, B. Fei, "Medical hyperspectral imaging: a review", *Journal of biomedical optics*, vol. 19, no. 1, 2014.

[54] B. Khoobehi, J.M. Beach, H. Kawano, "Hyperspectral imaging for measurement of oxygen saturation in the optic nerve head", *Investigative ophthalmology & visual science*, vol. 45, no. 5, pp. 1464-1472, 2004.

[55] D.C. Kellicut et al., "Emerging technology: hyperspectral imaging", *Perspectives in vascular surgery and endovascular therapy*, vol. 16, no. 1, pp. 53-57, 2004.

[56] J. Freeman et al., "Multispectral and hyperspectral imaging: applications for medical and surgical diagnostics," *in Proc. of the 19th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, Chicago, Illinois, pp. 700–701, 1997.

[57] S. T. Monteiro et al., "Towards applying hyperspectral imagery as an intraoperative visual aid tool," *in Proc. 4th Int. Conf. on Visualization, Imaging and Image Processing*, Marbella, Spain, pp. 483–488, 2004.

[58] S. V. Panasyuk et al., "Medical hyperspectral imaging to facilitate residual tumor identification during surgery," *Cancer Biol. Ther,* vol. 6, no. 3, pp. 439–446, 2007.

[59] B. Fei, H. Akbari, L.V. Halig, "Hyperspectral imaging and spectral-spatial classification for cancer detection", *Biomedical Engineering and Informatics (BMEI)*, 2012 5th International Conference on. IEEE, 2012.

[60] S. Kiyotoki et al., "New method for detection of gastric cancer by hyperspectral imaging: a pilot study", *Journal of biomedical optics*, vol. 18, no. 2, 2013.

[61] Z. Liu, H. Wang, Q. Li, "Tongue tumor detection in medical hyperspectral images", *Sensors*, vol. 12, no. 1, pp. 162-174, 2011.

[62] S. Saponara, A. Plaza, M. Diani, M.F. Carlsohn, G. Corsini, "Special issue on algorithms and architectures for real-time multi-dimensional image processing", *Journal of Real-Time Image Processing*, pp. 1-4, 2014.

[63] R. Lu, "Detection of bruises on apples using near-infrared hyperspectral imaging", *Transactions-American Society of Agricultural Engineers*, vol. 46, no. 2, pp. 523-530, 2003.

[64] L. S. Bernstein et al., "A new method for atmospheric correction and aerosol optical property retrieval for VIS-SWIR multi and hyperspectral imaging sensors: QUAC (QUick atmospheric correction)", *Proc. IEEE IGARSS*, vol. 5, pp. 3549 -3552, 2005.

[65] H. Akbari, Y. Kosugi, K. Kojima, N. Tanaka, "Detection and analysis of the intestinal ischemia using visible and invisible hyperspectral imaging", *Biomedical Engineering, IEEE Transactions* on, vol. 57, no. 8, pp. 2011-2017, 2011.

[66] H. Akbari, K. Uto, Y. Kosugi, K. Kojima, N. Tanaka, "Cancer detection using infrared hyperspectral imaging", *Cancer science*, vol. 102, no. 4, pp. 852-857, 2011.

[67] T. Cervero et al., "Scalable architectures for real-time hyperspectral unmixing", *Microelectronics Journal*, 2014.

[68] J.M.P. Nascimento, J.M. Bioucas-Dias, J.M. Rodriguez Alves, V. Silva, A. Plaza, "Parallel hyperspectral unmixing on GPUs", *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 666–670, 2014.

[69] A. Remón, S. Sánchez, A. Paz, E.S. Quintana-Ortí, A. Plaza, "Real-time endmember extraction on multicore processors", *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 5, pp. 924–928, 2011.

[70] S. Sánchez, R. Ramalho, L. Sousa, A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs", *Journal of Real-Time Image Processing*, pp. 1-15, 2012.

[71] C. Lucarz, I. Amer, M. Mattavelli, "Reconfigurable video coding: Objectives and technologies", 16th IEEE International Conference on Image Processing (ICIP), pp. 749-752, Nov. 2009.

[72] J. Eker and J. W. Janneck, "Cal language report", Tech. Rep. UCB/ERL M03/48, University of California at Berkeley, December 2003.

[73] S. Browne, J. Dongarra, G. Ho, P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors", International Journal of High Performance Computing Applications, vol. 14, no. 3, pp. 189 – 204, 2000.

[74] C. Chang, Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery", *IEEE Trans.Geosci. and Remote Sens.*, vol.42, no.3, pp.608–619, 2004.

[75] D. Heinz, C.-I Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery", *IEEE Trans. Geosci. and Remote Sens.*, vol. 39, pp. 529–545, 2001.

[76] M. Vélez, "Iterative Algorithms for Unmixing of Hyperspectral Imagery", *In Proceeding of SPIE*, 2003.

[77] C. González, J. Resano, A. Plaza, D. Mozos, "FPGA Implementation of abundance estimation for spectral unmixing of hyperspectral data using the Image Space Reconstruction Algorithm", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 1, pp. 248-261, 2012.

ANEXOS

ANEXO 1. MANUAL DE CONFIGURACIÓN E INSTALACIÓN

En el presente anexo se proporciona un tutorial de instalación y configuración del entorno necesario para la correcta utilización de la librería desarrollada en este Proyecto Fin de Grado. Se ha de resaltar que este tutorial se ha desarrollado y, por tanto, está probado para un ordenador con las siguientes características:

- Sistema operativo: Ubuntu 12.04 LTS
- Compilador: gcc versión 4.6.3
- Eclipse 4.3 (Kepler) y/o Eclipse 4.4 (Luna)
- ORCC versión 2.1.1
- Librería OTB versión 3.12.0
- Librería ITK versión 4.5.1
- Librería VXL versión 1.14.0

El tutorial se divide en dos partes:

- Tutorial de configuración e instalación de las librerías ITK, OTB y VXL. Este tutorial es necesario para poder hacer un correcto uso de la librería desarrollada.
- Tutorial de configuración e instalación de Eclipse y ORCC. Este tutorial sólo es necesario si se quiere comprobar que el proceso anterior se ha realizado con éxito, puesto que, para ello, se necesita compilar un proyecto ORCC. Por tanto, su seguimiento es opcional.

A.1.1. Librerías ITK, OTB y VXL

Los pasos a seguir son los siguientes:

1. Descargar el fichero comprimido *instalador.tar.gz*
2. Extraer el contenido en el directorio *Home*, tal y como se observa en la figura A.1.1.

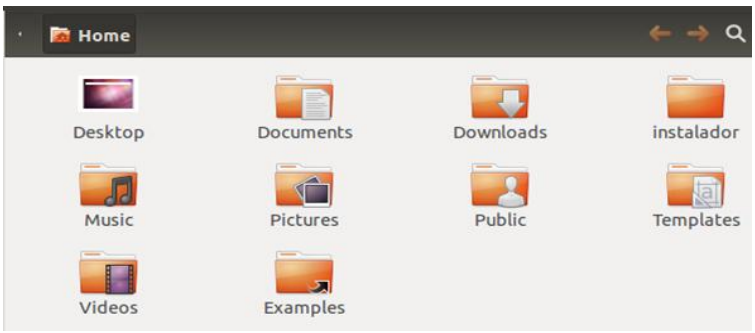


Fig. A.1.1. Contenido del directorio Home

El contenido de esta carpeta se puede observar en la figura A.1.2.

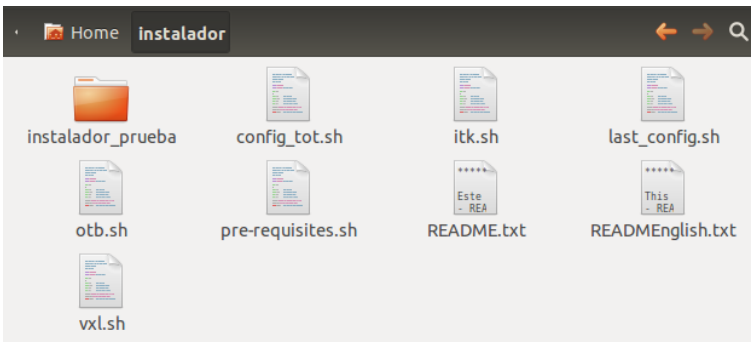


Fig. A.1.2. Contenido del directorio instalador

Como podemos observar en dicha figura, la carpeta *instalador* debe contener ocho ficheros y un directorio:

- Ficheros:
  - ✓ *pre-requisites.sh*
  - ✓ *itk.sh*
  - ✓ *otb.sh*
  - ✓ *vxl.sh*
  - ✓ *last_config.sh*
  - ✓ *config_tot.sh*
  - ✓ *README.txt*
  - ✓ *READMEnglish.txt*
- Directorio: *instalador_prueba*

De los ficheros, todos los que tienen la extensión *.sh* son ficheros de configuración, mientras que los *.txt* son ficheros de texto en los que se explica cómo usar dichos *.sh* -en español y en inglés-.
Respecto al directorio, éste aporta un proyecto ORCC - Eclipse para la comprobación de la correcta ejecución del proceso de instalación.

3. Abrir un terminal y navegar hasta la nueva carpeta – figura A.1.3 -.

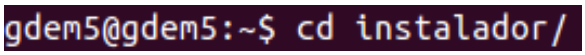


Fig. A.1.3. Navegación a la nueva carpeta

4. Ejecutar el primer fichero de instalación, escribiendo en el terminal el comando mostrado en la figura A.1.4.

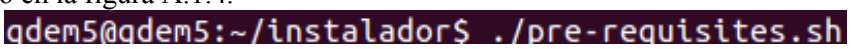


Fig. A.1.4. Ejecución del primer instalador

La duración de este instalador es variable pero, si tiene su ordenador actualizado, no tardará más de unos quince minutos.

Cuando su terminal tenga una salida como la de la figura A.1.5, se ha de introducir la contraseña de la sesión abierta en el PC.

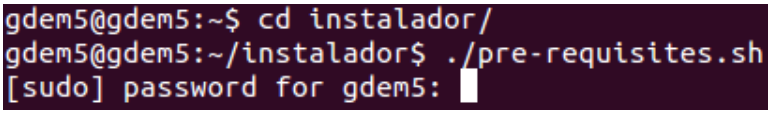


Fig. A.1.5. Contraseña del comando sudo

Asimismo, a lo largo de la instalación se solicitará la confirmación del usuario para ciertas operaciones, como puede observarse en la figura A.1.6.

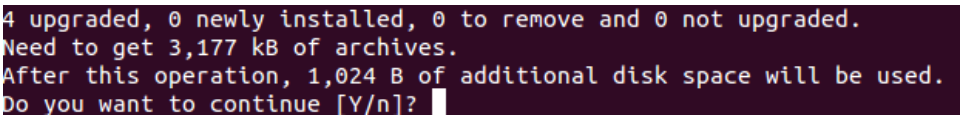


Fig. A.1.6. Confirmación de operación

Cuando esto ocurra, se ha de escribir *Y* en el terminal - o *S*, si el ordenador está configurado en español- y pulsar *Enter*.
Estas serán las únicas dos interacciones del usuario con el configurador, además de ejecutar cada fichero.

5. Esperar a que finalice el configurador anterior. Cuando lo haga, el terminal debe tener una salida parecida a la que aparece en la figura A.1.7.

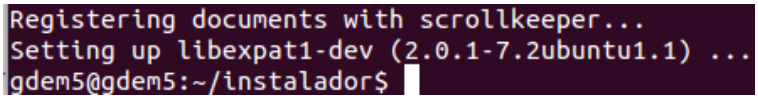


Fig. A.1.7. Finalización de pre-rquisites.sh

6. Ejecutar el siguiente fichero de instalación, escribiendo en el terminal el comando mostrado en la figura A.1.8.

```
gdem5@gdem5:~/instalador$ ./itk.sh
```

Fig. A.1.8. Ejecución del segundo instalador

Este configurador es el más complejo de todo el instalador, por lo que su duración puede llegar a extenderse hasta las tres horas.
Al igual que en el instalador anterior, la interacción del usuario con el proceso de instalación se reduce a introducir la contraseña del comando *sudo* y confirmar (*Y* / *S*) la instalación de paquetes extra.
Este instalador creará una nueva carpeta en el directorio *Home*, llamada *itk* (ver figura A.1.9), en la que se realizará la descarga y configuración de dicha librería.

Fig. A.1.9. Creación de la carpeta itk

Como hemos mencionado anteriormente, la compilación de la librería ITK puede llegar a extenderse hasta las tres horas, por lo que, en cuanto comience el proceso de compilación, el usuario puede dejar de prestar atención a este instalador. En concreto, la compilación alcanza el 60% rápidamente -en escasos minutos- pero, a partir de ahí, el proceso se ralentiza. Por tanto, el usuario puede dejar de prestar toda su atención al instalador cuando su proceso alcance un nivel similar al mostrado en la figura A.1.10.

```
inking CXX static library ../../../../lib/libITKSpatialObjects-4.5.a
 61%] Built target ITKSpatialObjects
 61%] Generating test/ITKQuadEdgeMeshHeaderTest1.cxx
canning dependencies of target ITKQuadEdgeMeshHeaderTest1
 61%] Building CXX object Modules/Core/QuadEdgeMesh/CMakeFiles/ITKQuadEdgeMesh
inking CXX executable ../../../bin/ITKQuadEdgeMeshHeaderTest1
 61%] Built target ITKQuadEdgeMeshHeaderTest1
canning dependencies of target ITKQuadEdgeMeshTestDriver
 61%] Building CXX object Modules/Core/QuadEdgeMesh/test/CMakeFiles/ITKQuadEdg
```

Fig. A.1.10. Proceso de compilación de la librería ITK

7. Esperar a que la instalación de la librería ITK finalice. Cuando lo haga, el terminal debería mostrar una salida similar a la mostrada en la figura A.1.11.

```
-- Installing: /usr/local/include/ITK-4.5/itkFrameAverageVideoFilter.hxx
-- Installing: /usr/local/include/ITK-4.5/itkFrameDifferenceVideoFilter.h
-- Installing: /usr/local/include/ITK-4.5/itkImageFilterToVideoFilterWrapper.hxx
-- Installing: /usr/local/include/ITK-4.5/itkFrameDifferenceVideoFilter.hxx
-- Installing: /usr/local/include/ITK-4.5/itkDecimateFramesVideoFilter.h
-- Installing: /usr/local/include/ITK-4.5/itkImageFilterToVideoFilterWrapper.h
-- Installing: /usr/local/include/ITK-4.5/itkFrameAverageVideoFilter.h
-- Installing: /usr/local/include/ITK-4.5/itkDecimateFramesVideoFilter.hxx
-- Installing: /usr/local/lib/cmake/ITK-4.5/Modules/ITKVideoFiltering.cmake
gdem5@gdem5:~/instalador$
```

Fig. A.1.11. Finalización de itk.sh

8. Ejecutar el comando que aparece en la figura A.1.12 para instalar la librería OTB. La duración aproximada de este instalador es de unos quince minutos.

```
gdem5@gdem5:~/instalador$ ./otb.sh
```

Fig. A.1.12. Ejecución del tercer instalador

Al igual que en los instaladores anteriores, la interacción del usuario con el proceso de instalación se limita a introducir la contraseña del comando *sudo* y confirmar la instalación de paquetes extra.
Este instalador configura la librería directamente en las carpetas del sistema, por lo que no se crea ninguna carpeta adicional en el directorio *Home*.

9. Esperar a que termine el instalador de la librería OTB. Cuando el instalador finalice, en la ventana del terminal debería aparecer una salida similar a la de la figura A.1.13.

Fig. A.1.13. Finalización de otb.sh

10. Ejecutar en el terminal el comando que se muestra en la figura A.1.14.

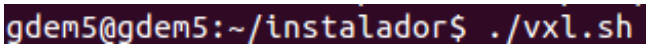


Fig. A.1.14. Ejecución del cuarto instalador

Este instalador tendrá una duración aproximada de 40 minutos. Por ello, una vez comenzada la compilación (ver figura A.1.15), el usuario no necesitará dedicar su atención a la misma, ya que su intervención no será necesaria.

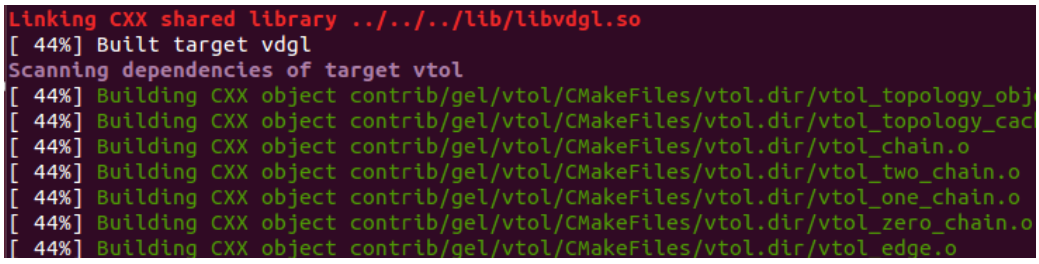


Fig. A.1.15. Proceso de compilación de la librería VXL

Cabe destacar que, al igual que con la librería ITK, el instalador creará una nueva carpeta en el directorio *Home*, tal y como se puede observar en la figura A.1.16. En esta carpeta será donde se configurará e instalará la librería VXL.

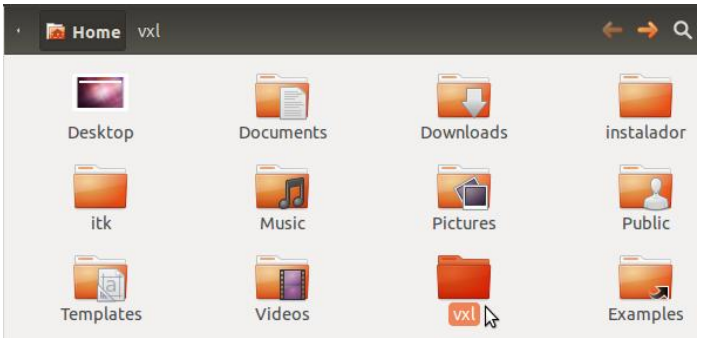


Fig. A.1.16. Creación de la carpeta vxl

11. Esperar a que finalice la instalación de la librería VXL. Cuando finalice, el terminal debería tener una salida similar a la observada en la figura A.1.17:

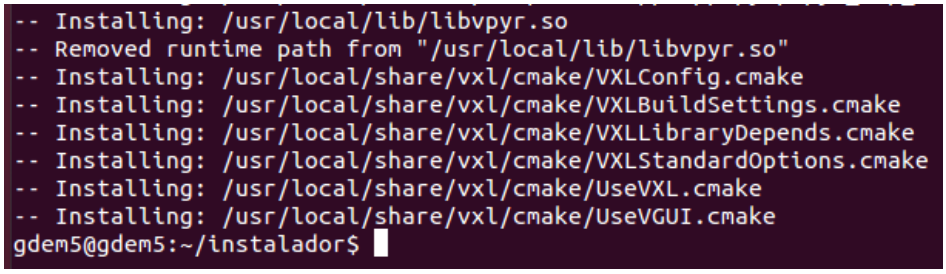


Fig. A.1.17. Finalización de vxl.sh

12. Ejecutar en el terminal el comando mostrado en la figura A.1.18.

```
gdem5@gdem5:~/instalador$ ./last_config.sh
```

Fig. A.1.18. Ejecución del último instalador

Este instalador realizará las últimas configuraciones, y su ejecución es prácticamente instantánea. No obstante, cabe destacar que una de las acciones realizadas por este instalador es la modificación del fichero *.profile* y, para que estos cambios sean efectivos, es necesario reiniciar el sistema, por lo que el instalador finaliza con una instrucción para reiniciar el sistema, tal y como puede observarse en la figura A.1.19. Por tanto, no olvide cerrar y guardar todos sus documentos antes de ejecutar este instalador, pues su ordenador se reiniciará automáticamente.

```
# Rebooting system -> Apply changes in .profile
sudo reboot
```

Fig. A.1.19. Última instrucción de last_config.sh

Una vez reiniciado el sistema, la instalación ya está completa. Para comprobar que todo se ha realizado correctamente, en la carpeta del instalador se proporciona un proyecto ORCC – Eclipse[22]. Si este proyecto compila y se ejecuta, la instalación se habrá realizado correctamente. Para ello, a continuación proporcionamos un tutorial de instalación y prueba de este sistema. Como hemos mencionado al inicio de este documento, este tutorial es opcional.

Por último, antes de finalizar este tutorial, hemos de mencionar que, además de este proceso de configuración, también se proporciona un configurador alternativo, en caso de que el usuario desee reducir su interacción con el proceso al mínimo. Para ello, en lugar de ejecutar los cinco instaladores anteriores, el usuario ejecutaría únicamente uno, que recibe el nombre de *config_tot.sh* (ver figura A.1.20). Este instalador ejecuta secuencialmente los cinco configuradores utilizados anteriormente (ver figura A.1.21), de tal forma que, en este caso, la interacción del usuario con el proceso se reduce a introducir la contraseña del comando *sudo* y a aceptar la instalación de paquetes extra.

Sin embargo, hemos de destacar que no recomendamos la utilización de este instalador, puesto que su duración podría llegar a extenderse más allá de las cinco horas y, además, en caso de fallo, la localización del mismo es mucho más sencilla con el otro procedimiento.

```
gdem9@GDEM9:~/instalador$ ./config_tot.sh
```

Fig. A.1.20. Ejecución del configurador global

config_tot.sh

```
#########################################################################
#                                                                       #
#                     Fichero de configuración total                    #
#                                                                       #
#  Se recomienda no utilizar este fichero:                              #
#  Su ejecución puede llegar a extenderse durante más de cinco horas    #
#                                                                       #
#########################################################################

sh ./pre-requisites.sh
sh ./itk.sh
sh ./otb.sh
sh ./vxl.sh
sh ./last_config.sh
```

Fig. A.1.21. Contenido de config_tot.sh

[22] El proyecto proporcionado contiene ya el código autogenerado por Orcc para Eclipse Luna. Si usted está utilizando esta versión, puede saltarse directamente al último paso de este tutorial. Si no es su caso, deberá ejecutar el tutorial completo.

A.1.2. Eclipse y ORCC

Instalación de Eclipse

1. Descargar el entorno Eclipse de la página oficial: http://www.eclipse.org/downloads/
   En concreto, se descargará la versión *Eclipse IDE for C/C++ Developers*.
2. Extraer el fichero comprimido descargado en el directorio que estime oportuno -por ejemplo, en *Home* o en el Escritorio.
   Nota: Durante el desarrollo de este proyecto, Eclipse ha lanzado una nueva versión (Eclipse 4.4 Luna). Por ello, este tutorial da soporte tanto a la versión antigua (Eclipse 4.3 Kepler) como a la nueva (Eclipse 4.4 Luna). El único cambio detectado entre ambas versiones se localiza en el *CMakeLists.txt* que genera ORCC al compilar un proyecto, por lo que aportaremos ambos para que el usuario utilice el que resulte necesario.
3. Descargar Java de la página oficial:
   https://www.java.com/es/download/linux_manual.jsp?locale=es
   Extraer el fichero comprimido descargado y renombrar la carpeta resultante con el nombre *jre*, como puede apreciarse en la figura A.1.22.

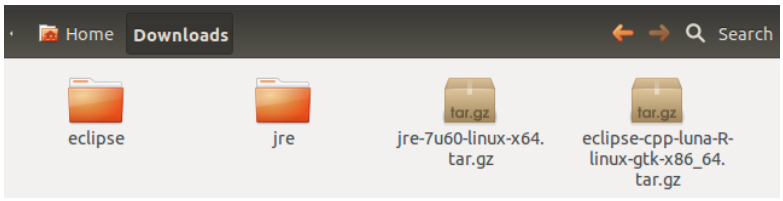


Fig. A.1.22. Descarga de Java

4. Copiar la carpeta *jre* obtenida en el paso anterior dentro de la carpeta Eclipse, como se muestra en la figura A.1.23.

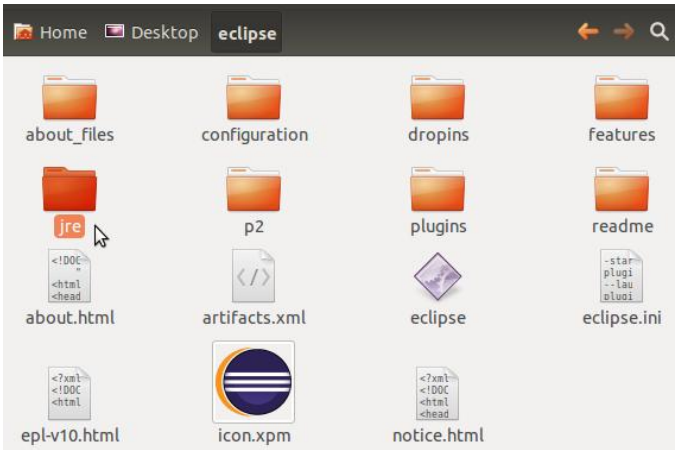


Fig. A.1.23. Inserción de jre *en la carpeta* eclipse

Instalación de ORCC

1. Abrir Eclipse y seleccionar el menú *Help* → *Install new software...* → *Add...*
   En *Location*, añadir *http://orcc.sourceforge.net/eclipse* (ver figura A.1.24).

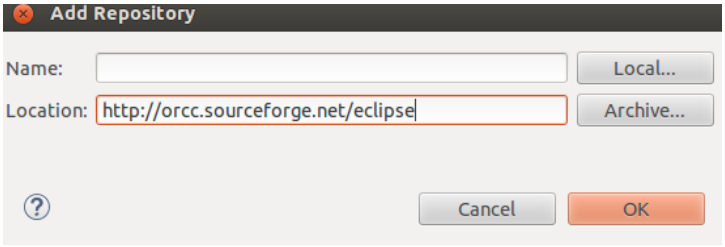


Fig. A.1.24. Ventana de descarga del software ORCC

2. Seleccionar *Open RVC-Cal Compiler* y pulsar *Next* → *Next* (ver figura A.1.25).

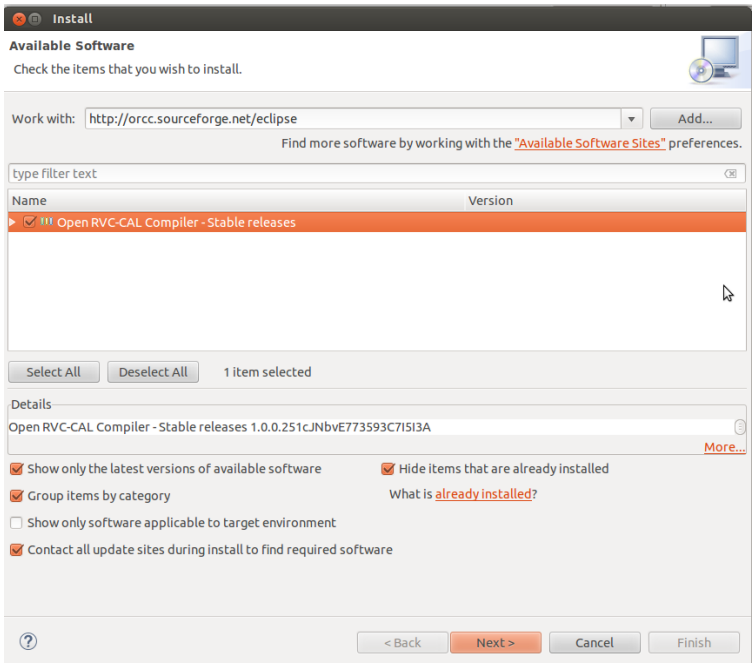


Fig. A.1.25. Selección de RVC – CAL compiler

3. Aceptar la licencia y finalizar, como se muestra en la figura A.1.26.

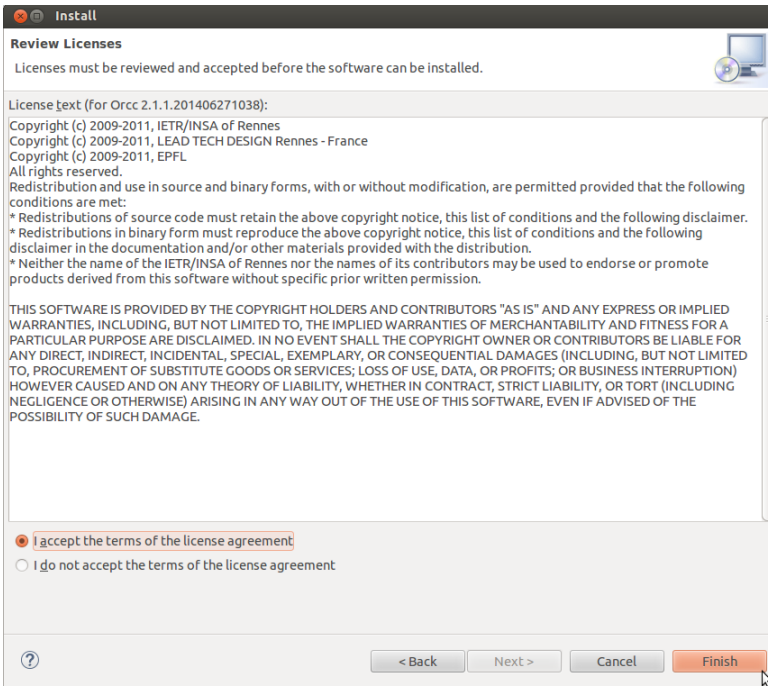


Fig. A.1.26. Licencia de ORCC 2.1.1

4. Darle a Ok cuando aparezca el mensaje que se muestra en la figura A.1.27.

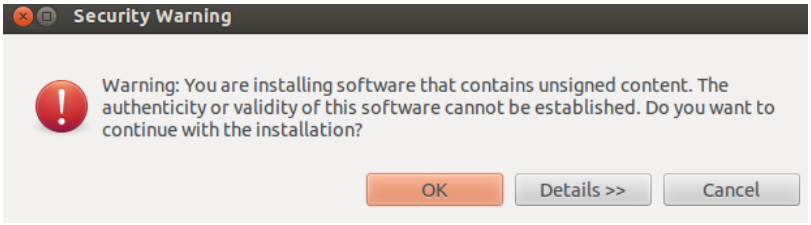


Fig. A.1.27. Mensaje de aviso

5. Reiniciar Eclipse. Si existe algún problema durante esta configuración, consultar la siguiente página: *http://orcc.sourceforge.net/getting-started/install-orcc/*
6. Cuando solicite la ruta al *workspace*, seleccionar la carpeta *project* dentro del directorio *instalador_prueba*. La ruta completa es *instalador/instalador_prueba/project*.
7. Pulsar en *File* → *Import* → *General* → *Existing Projects into Workspace* → *Next*
8. En *root directory*, seleccionar la ruta a la carpeta *hsi_system* que se proporcionaba dentro del instalador. En concreto, la ruta a la carpeta es *instalador/instalador_prueba/project/hsi_system.*
9. En *Projects*, seleccionar el proyecto a importar y seleccionar *Finish*, tal y como se muestra en la figura A.1.28.

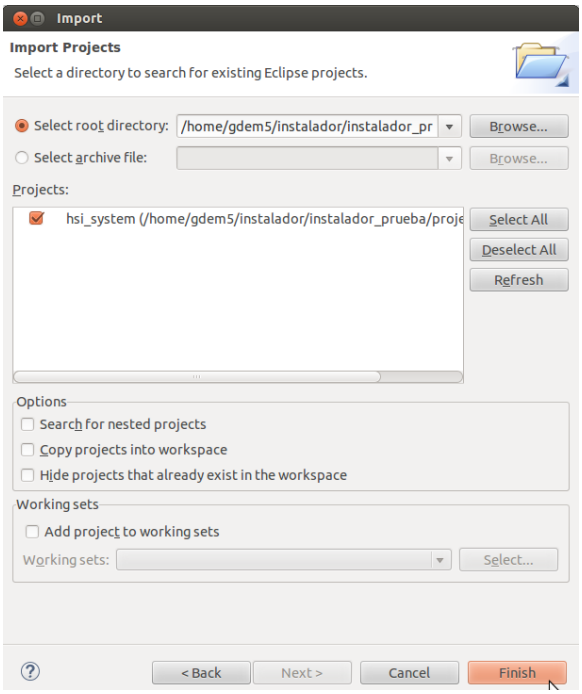


Fig. A.1.28. Ventana de importación de proyectos

10. En el proyecto, navegar hasta *hsi_system.xdf*, seleccionarla y pulsar botón derecho *Run as...* → *Run Configurations* → *ORCC compilation* y configurar la compilación con los parámetros mostrados en la figura A.1.29.

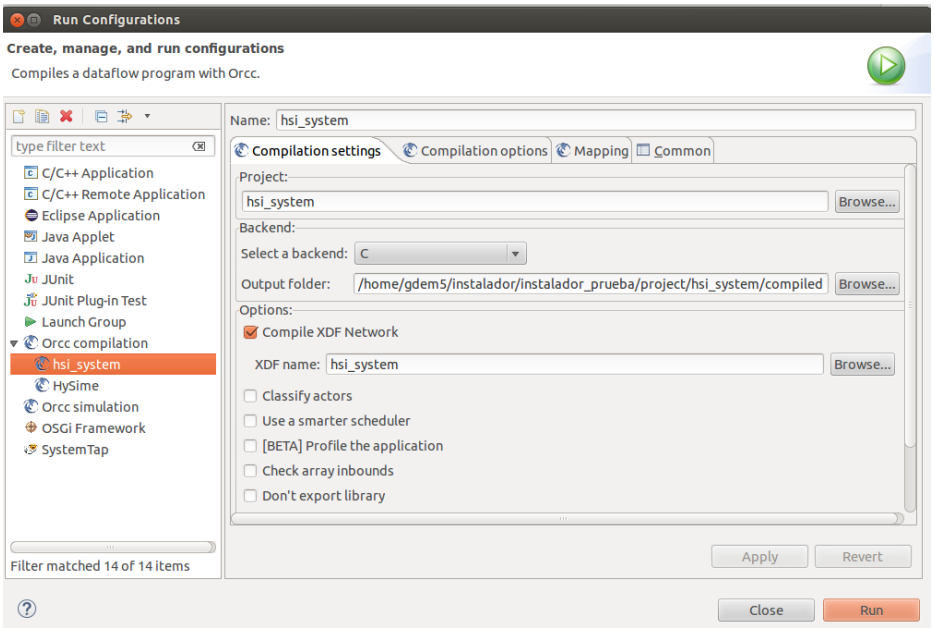


Fig. A.1.29. Configuración de los parámetros de compilación

11. Pulsar *Run*. En la consola debe aparecer un mensaje similar al de la figura A.1.30.

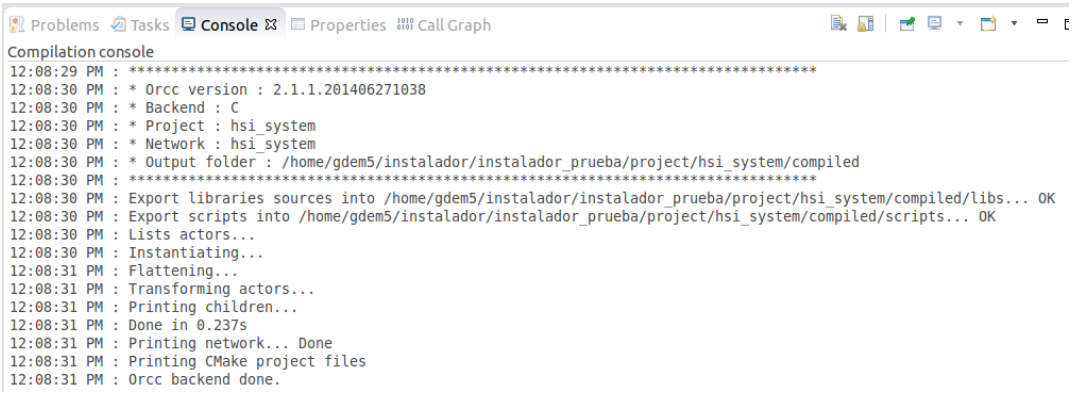


Fig. A.1.30. Resultado de la compilación en ORCC

12. Abrir un terminal y navegar hasta la carpeta *instalador_prueba*, que está dentro del directorio instalador. El contenido de esta carpeta se puede observar en la figura A.1.31.

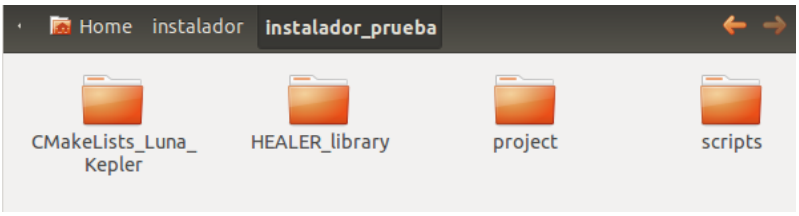


Fig. A.1.31. Contenido de la carpeta instalador_prueba

Como podemos observar, la carpeta *instalador_prueba* contiene cuatro directorios:

- *CMakeLists_Luna_Kepler*: En este directorio aportamos los archivos necesarios para garantizar la compatibilidad del sistema con ambas versiones de Eclipse -Luna y Kepler-, para que, a la hora de la compilación, se emplee el fichero correspondiente a la versión que se esté utilizando.
- *HEALER library*: En este directorio se encuentran todos los ficheros fuente necesarios para la implementación de la librería desarrollada en este Proyecto Fin de Grado, y que serán requeridos durante el proceso de compilación.
- *project*: En este directorio se proporciona un proyecto Eclipse - ORCC cuyo objetivo es la comprobación de la correcta realización de este tutorial. Además, se proporciona un *workspace* ya construido (versión Eclipse Luna) para poder comprobar de forma sencilla el correcto funcionamiento del proyecto.
- *scripts*: En este directorio se proporcionan los ficheros de configuración necesarios para automatizar la compilación del programa. En concreto, se proporcionan dos ficheros: *compiler_kepler.sh* y *compiler_luna.sh*. Cuando se vaya a compilar el proyecto, el usuario únicamente tendrá que navegar hasta esta carpeta y ejecutar el *compiler* adecuado a su versión de Eclipse.

  Este tutorial se ha desarrollado con *Eclipse Luna*, por lo que mostramos la ejecución del proceso de compilación para esta versión, como puede observarse en la figura A.1.32.

13. En el terminal, ejecutar uno de los siguientes comandos:

    ./compiler-kepler.sh si su versión es *Eclipse Kepler*

    ./compiler-luna.sh si su versión es *Eclipse Luna*

```
gdem5@gdem5:~$ cd instalador/instalador_prueba/
gdem5@gdem5:~/instalador/instalador_prueba$ cd scripts/
gdem5@gdem5:~/instalador/instalador_prueba/scripts$ ./compiler_luna.sh
```

Fig. A.1.32. Ejecución de compiler-luna.sh

Este fichero realiza la compilación del proyecto y, si todo se realiza de forma correcta, ejecuta la aplicación generada al finalizar la compilación. Si todo el proceso se ha realizado de forma satisfactoria, la salida de su terminal debería ser parecida a la mostrada en la figura A.1.33.

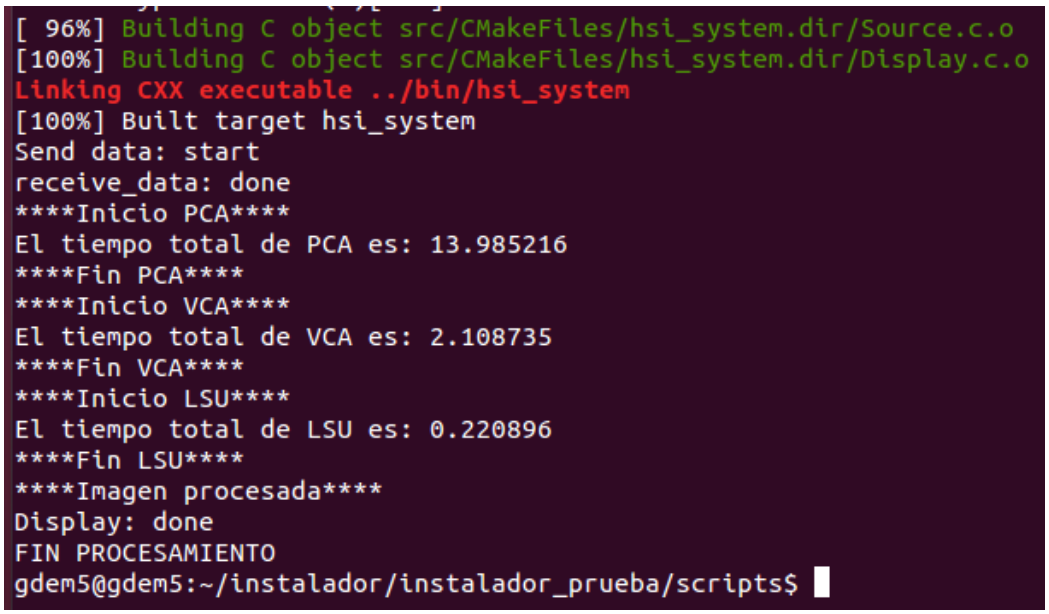


Fig. A.1.33. Ejecución de la aplicación

Éste es el final del manual de instalación y configuración del entorno para la utilización de la librería desarrollada. Si su salida final ha sido similar a la mostrada en la figura A.1.33, el proceso se ha realizado con éxito y ya puede utilizar la librería en su ordenador.

ANEXO 2. MANUAL DE USUARIO: API DE LA LIBRERÍA

En el presente anexo se proporciona un manual de usuario de la librería desarrollada en este Proyecto Fin de Grado. Este manual está compuesto de un índice de las funciones que la componen y una descripción detallada de cada una de ellas, aportando una breve descripción de su funcionalidad y explicando la naturaleza de los parámetros de entrada y salida. Asimismo, también se proporciona un índice de las librerías que se necesitan y que, en consecuencia, se importan en esta librería.
Este documento constituye el API de la librería, que también se proporciona en formato digital, tanto en PDF como en HTML. Cabe destacar que, debido a la naturaleza del proyecto en el que se inscribe esta librería, la documentación está en inglés.

A.2.1. Librerías necesarias

En esta sección, se proporciona una relación de las librerías requeridas por las funciones desarrolladas:

#include <assert.h>

#include <dlfcn.h>

#include <dirent.h>

#include <errno.h>

#include <fcntl.h>

#include <malloc.h>

#include <math.h>

#include <memory.h>

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <time.h>

#include <unistd.h>

#include "cppWrapper.h"

Como se puede observar, excepto *cppWrapper.h*, todas las librerías son librerías de sistema, por lo que no se requiere ninguna descarga adicional. Respecto a *cppWrapper.h*, ésta se proporciona con el proyecto, y debe ser incluida en la carpeta *compiled/libs/orcc-native/include*. No obstante, este proceso está automatizado, por lo que no es necesaria la intervención del usuario (ver Anexo 3).

A.2.2. Índice de funciones

En esta sección se proporciona una relación de los prototipos de las funciones implementadas, así como una breve descripción de cada una de ellas:

float **double_to_float** (double *value*)
Cast between float and double types.

int **double_to_int** (double *value*)
Cast between int and double types.

double **elev** (double *base*, double *power*)
Raise a number to a given power.

void **end_program** ( )
Finish program execution.

double **float_to_double** (float *value*)
Cast between float and double types.

void **gen_conf_data** (int *rows*, int *columns*, int *bands*, int *nbits*, int *nformat*, int *numPC*, int *numEnd*)
Generate the Data.cal file.

void **gen_hdr** (double* *matrixOut*, int *rows*, int *columns*, int *lines*, int *samples*, int *bands*, int *numEnd*, char* *path*, char* *fileName*, int *typeFile*)
Generate a complete hyperspectral image, which has two files: a binary file and a .hdr file.

void **get_diagonal** (double* *matrix*, double* *vectorDiag*, int *positions*)
Get the diagonal vector of a matrix.

void **get_eigenvectMatrix** (double* *matrixIn*, double* *matrixEigenvectors*, int *rows*, int *columns*)
Get the eigenvectors matrix.

void **get_inverse** (double* *matrixIn*, double* *matrix_inverse*, int *rows*, int *columns*)
Get the inverse matrix.

void **get_pinverse** (double* *matrixIn*, double* *matrix_p_inverse*, int *rows*, int *columns*)
Get a pseudo - inverse matrix.

double **get_time** ( )
Get the real time of the system (in seconds, with a resolution of nanoseconds)

int **getIndexMax** (double* *vector*, int *positions*)
Get the position of the largest element vector (absolute values only)

double **int_to_double** (int *value*)
Cast between int and double types.

double **log** (double *value*)
Get the logarithm of a number (base 10): result = log(value)

void **matrix_extend** (double* *matrix1,* double* *matrix2,* double *rows1,* double *columns1,* double *rows2,* double *columns2*)
Extend a matrix.

void **matrix_fill_column** (double* *matrix1,* double *rows1,* double *columns1,* double *column*, double *value*)
Set a column of the matrix to the same value.

void **matrix_get_column** (double* *matrix1,* double* *vector*, double *rows1,* double *columns1,* double *column*)
Get a certain column from a matrix.

void **matrix_get_row** (double* *matrix1,* double* *vector*, double *rows1,* double *columns1,* int *row*)

Get a certain row from a matrix.

void **matrix_minus_matrix** (double* *matrix1,* double* *matrix2,* double* *matrixOut*, int *rows*, int *columns*)

Deduct two matrices.

void **matrix_minus_value** (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Deduct the same value to all the elements of a matrix.

void **matrix_mult** (double* *matrix1,* double* *matrix2,* double* *matrixResult*, double *rows1,* double *columns1,* double *rows2,* double *columns2*)

Multiply two matrices.

void **matrix_mult_vector** (double* *matrix1,* double* *vector*, double* *vectorResult*, double *rows1,* double *columns1,* double *positions_vector*)

Multiply a matrix by a vector (the result is a vector)

void **matrix_plus_matrix** (double* *matrix1,* double* *matrix2,* double* *matrixOut*, int *rows*, int *columns*)

Add two matrices.

void **matrix_plus_value** (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Add the same value to all the elements of a matrix.

void **matrix_reduce** (double* *matrix1,* double* *matrix2,* double *rows1,* double *columns1,* double *rows2,* double *columns2*)

Reduce matrix dimensions.

void **matrix_scale** (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Divide each element of a matrix by the same value.

void **matrix_set_column** (double* *matrix1,* double* *vector*, double *rows1,* double *columns1,* double *column*)

Set a certain column of a matrix.

void **matrix_set_row** (double* *matrix1,* double* *vector*, double *rows1,* double *columns1,* double *row*)

Set a certain row of a matrix.

double **maxSqrtSum** (double* matrix, double rows, double columns*)*

Square each element of each row and add them, row by row.

double **mean_vector** (double* *vector*, int *positions*)

Get the mean value of a given vector.

void **modify_config** (int *num_PC*, int *num_Endmembers*, char* *pathIn*, char* *var1,* char* *var2*)

Modify analysis chain configuration parameters.

void **random_vector** (double* *vector*, double *positions*)

Fill a vector with random values between 0 and 1 (of double type)

void **read_txt(**double* matrixIn, int rows, int columns, char* *fileName)*

Read a .txt file and save its contents in a matrix.

int **rem** (double *dividend*, int *divisor*)

Return the remainder of a division.

double **sq_root** (double *value*)

Get the square root of a number.

void **transpose**(double* *matrixIn*, double* *matrixTranspose*, double *rows*, double *columns*)
Transpose a matrix.

void **vector_minus_value** (double* *vectorIn,* double *value*, double* *vectorMinus,* int *positions*)
Subtract the same value from each position of a vector.

void **vector_minus_vector** (double* *vector1,* double* *vector2,* double* *vectorResult,* int *positions*)
Subtract one vector from another.

double **vector_mult** (double* *vector1,* double* *vector2,* int *positions*)
Multiply two vectors.

void **vector_mult_matrix** (double* *vector,* double* *matrix,* double* *vectorResult*, double *positions_vector*, double *rows1,* double *columns1*)
Multiply a matrix by a vector.

void **vector_scale** (double* *vectorIn*, double* *vectorOut*, double *value*, int *positions*)
Divide each element of a vector by the same value.

void **write_double_binary**(double* *matrixOut*, int *rows*, int *columns*, char* *fileName*)
Write an image into a binary file, with the BSQ format of an hyperspectral image.

void **write_short_binary(short** int* matrixOut, int rows, int columns, char* *fileName)*
Write an image into a binary file, with the BIP format of an hyperspectral image.

void **write_txt(**double* matrixOut, int rows, int columns, char* *fileName)*
Save a matrix into a .txt file.

A.2.3. Descripción detallada

En esta sección se proporciona una descripción detallada de cada una de las funciones enumeradas en la sección anterior, describiendo tanto su funcionalidad como los parámetros de entrada y salida.

1.1. float double_to_float (double *value*)
Cast between float and double types.

Parameters:

value	double value

Returns:
The same value, but as a float

1.2. int double_to_int (double *value*)
Cast between int and double types.

Parameters:

value	double value

Returns:
The same value, but as an int

1.3. double elev (double *base*, double *power*)
Raise a number to a given power.

Parameters:

base	Number we want to raise to a certain power
power	Power

Returns:
Result

1.4. void end_program ( )

Finish program execution

1.5. double float_to_double (float *value*)

Cast between float and double types.

Parameters:

value	float value

Returns:

The same value, but as a double

1.6. void gen_conf_data (int *rows*, int *columns*, int *bands*, int *nbits*, int *nformat*, int *numPC*, int *numEnd*)

Generate the Data.cal file.

This function is only used to generate the output of algorithms related to endmember estimation.

Parameters:

rows	Number of rows of the original hyperspectral image
columns	Number of columns of the original hyperspectral image
bands	Number of bands of the original hyperspectral image
nbits	Number of bits of each pixel
nformat	Number to select the format: 0 - BSQ, 1 - BIP, 2 - BIL
numPC	Number of principal components (PCA algorithm)
numEnd	Number of endmembers

1.7. void gen_hdr (double* *matrixOut*, int *rows*, int *columns*, int *lines*, int *samples*, int *bands*, int *numEnd*, char* *path*, char* *fileName*, int *typeFile*)

Generate a complete hyperspectral image, which has two files: a binary file and a .hdr file.

The last one is the header of the binary file, and it must have the same name, but with the .hdr extension. The file is written (row by row) in the bin directory of the project. The format of the analyzed hyperspectral images is supposed to be BSQ.

Parameters:

matrixOut	Image to save in the binary file
rows	Number of rows of *matrixOut*
columns	Number of columns of *matrixOut*
lines	Number of lines of the initial image
samples	Number of samples of the initial image
bands	Number of bands of the output hyperspectral image
numEnd	Number of endmembers of the output hyperspectral image
path	Path to the hyperspectral image associated with the .hdr generated
fileName	Name of the binary file (without the extension)
typeFile	Number to select the file we want to generate: 1 – Dimensionality reduction, 2 – Endmembers, 3 - Abundances

1.8. void get_diagonal (double* *matrix*, double* *vectorDiag*, int *positions*)

Get the diagonal vector of a matrix

Parameters:

matrix	Matrix whose diagonal we want to obtain (it has to be a square matrix)
vectorDiag	Diagonal vector (result)
positions	Number of rows or columns of *matrix*

1.9. void get_eigenvectMatrix (double* *matrixIn*, double* *matrixEigenvectors*, int *rows*, int *columns*)

Get the eigenvectors matrix

Parameters:

matrixIn	Matrix whose eigenvectors we want to obtain (it has to be a square matrix)
matrixEigenvectors	Eigenvectors matrix (result)
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.10. void get_inverse (double* *matrixIn*, double* *matrix_inverse*, int *rows*, int *columns*)

Get the eigenvectors matrix

Parameters:

matrixIn	Matrix whose inverse we want to obtain (it has to be a square matrix)
matrix_inverse	Inverse matrix (result)
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.11. void get_pinverse (double* *matrixIn*, double* *matrix_p_inverse*, int *rows*, int *columns*)

Get the eigenvectors matrix

Parameters:

matrixIn	Matrix whose pseudo - inverse we want to obtain (it has to be a square matrix)
matrix_p_inverse	Pseudo - inverse matrix (result)
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.12. double get_time ( )

Get the real time of the system (in seconds, with a resolution of nanoseconds)

Returns:

System time

1.13. int getIndexMax (double* *vector*, int *positions*)

Get the position of the largest element vector (absolute values only)

Parameters:

vector	Vector to use in the operation
positions	Number of elements of *vector*

Returns:

Returns the position (index) of the largest element vector

1.14. double int_to_double (int *value*)

Cast between int and double types.

Parameters:

value	int value

Returns:

The same value, but as a double

1.15. double log (double *value*)

Get the logarithm of a number (base 10): result = log(value)

Parameters:

value	Number whose logarithm we want to calculate

Returns:

Result

1.16. void matrix_extend (double* *matrix1*, double* *matrix2*, double *rows1*, double *columns1*, double *rows2*, double *columns2*)

Extend a matrix.

Parameters:

matrix1	Matrix whose dimensions we want to extend
matrix2	Extended matrix (result)
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
rows2	Number of rows of *matrix2*
columns2	Number of columns of *matrix2*

1.17. void matrix_fill_column (double* *matrix1*, double *rows1*, double *columns1*, double *column*, double *value*)

Set each element of a column of the input matrix to the same value.

Parameters:

matrix1	Matrix whose column we want to fill
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
column	Position of the filled column
value	Value of each element of the filled column

1.18. void matrix_get_column (double* *matrix1*, double* *vector*, double *rows1*, double *columns1*, double *column*)

Get a certain column from a matrix.

Parameters:

matrix1	Matrix whose column we want to obtain
vector	Vector where the required column is saved
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
column	Column number

1.19. void matrix_get_row (double* *matrix1*, double* *vector*, double *rows1*, double *columns1*, int *row*)

Get a certain row from a matrix.

Parameters:

matrix1	Matrix whose row we want to obtain
vector	Vector where the required row is saved
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
row	Row number

1.20. void matrix_minus_matrix (double* *matrix1*, double* *matrix2*, double* *matrixOut*, int *rows*, int *columns*)

Deduct two matrices.

Parameters:

matrix1	Minuend
matrix2	Subtrahend
matrixOut	Matrix where the result is saved
rows	Number of rows of *matrix1*and *matrix2*
columns	Number of columns of *matrix1*and *matrix2*

1.21. void matrix_minus_value (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Deduct the same value to all the elements of a matrix.

Parameters:

matrixIn	Matrix whose elements we want to edit
matrixOut	Matrix where the result is saved
value	Value to deduct
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.22. void matrix_mult (double* *matrix1*, double* *matrix2*, double* *matrixResult*, double *rows1*, double *columns1*, double *rows2*, double *columns2*)

Multiply two matrices.

Parameters:

matrix1	First factor
matrix2	Second factor
matrixResult	Product
rows1	Number of rows of first factor
columns1	Number of columns of first factor
rows2	Number of rows of second factor
columns2	Number of columns of second factor

1.23. void matrix_mult_vector (double* *matrix1*, double* *vector*, double* *vectorResult*, double *rows1*, double *columns1*, double *positions_vector*)

Multiply a matrix by a vector (the result is a vector)

Parameters:

matrix1	First factor
vector	Second factor
vectorResult	Product
rows1	Number of rows of first factor
columns1	Number of columns of first factor
positions_vector	Number of elements of second factor

1.24. void matrix_plus_matrix (double* *matrix1*, double* *matrix2*, double* *matrixOut*, int *rows*, int *columns*)

Add two matrices.

Parameters:

matrix1	First operand
matrix2	Second operand
matrixOut	Matrix where the result is saved
rows	Number of rows of *matrix1* and *matrix2*
columns	Number of columns of *matrix1* and *matrix2*

1.25. void matrix_plus_value (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Add the same value to all the elements of a matrix.

Parameters:

matrixIn	Matrix whose elements we want to edit
matrixOut	Matrix where the result is saved
value	Value to add
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.26. void matrix_reduce (double* *matrix1*, double* *matrix2*, double *rows1*, double *columns1*, double *rows2*, double *columns2*)

Reduce matrix dimensions.

Parameters:

Matrix1	Matrix whose dimensions we want to reduce
matrix2	Reduced matrix (result)
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
rows2	Number of rows of *matrix2*
columns2	Number of columns of *matrix2*

1.27. void matrix_scale (double* *matrixIn*, double* *matrixOut*, double *value*, int *rows*, int *columns*)

Divide each element of a matrix by the same value.

Parameters:

matrixIn	Each element of this matrix is the dividend of each operation
matrixOut	Result of the division
value	Divider of each operation
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.28. void matrix_set_column (double* *matrix1*, double* *vector*, double *rows1*, double *columns1*, double *column*)

Set a certain column of a matrix.

Parameters:

matrix1	Matrix whose column we want to modify
vector	Contains the new values of the column we want to edit
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
column	Column number

1.29. void matrix_set_row (double* *matrix1*, double* *vector*, double *rows1*, double *columns1*, double *row*)

Set a certain row of a matrix.

Parameters:

matrix1	Matrix whose row we want to modify
vector	Contains the new values of the row we want to edit
rows1	Number of rows of *matrix1*
columns1	Number of columns of *matrix1*
row	Row number

1.30. double maxSqrtSum (double* *matrix*, double *rows*, double *columns*)

Square each element of each row and add them, row by row.

Parameters:

matrix	Matrix to use in the operation
rows	Number of rows of *matrix*
columns	Number of columns of *matrix*

Returns:

The function returns the square root of the largest row sum of squares

1.31. double mean_vector (double* *vector*, int *positions*)

Get the mean value of a given vector.

Parameters:

vector	Vector whose mean value we want to obtain
positions	Size of *vector*

Returns:

Mean value

1.32. void modify_config (int *num_PC*, int *num_Endmembers*, char* *pathIn*, char* *var1*, char* *var2*)

Modify analysis chain configuration parameters.

Parameters:

num_PC	Number of principal components we want to obtain
num_Endmembers	Number of *endmembers* we want to obtain
pathIn	Path to the generated code of the HSI analysis chain
var1	Name of the first variable we want to modify (number of principal components)
var2	Name of the second variable we want to modify (number of *endmembers*)

1.33. void random_vector (double* *vector*, double *positions*)

Fill a vector with random values between 0 and 1 (of double type)

Parameters:

vector	Vector filled with random values
positions	Number of elements of *vector*

1.34. void read_txt(double* *matrixIn*, int *rows*, int *columns*, char* *fileName*)

Read a .txt file (row by row) and save its contents in a matrix.

Hence, the .txt file should contain only a matrix, and it should be located in the folder where the executable file is generated

Parameters:

matrixIn	Matrix where the result is going to be saved
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*
fileName	Name of the required .txt file. The extension must also be written in this parameter

1.35. int rem (double *dividend*, int *divisor*)

Return the remainder of a division.

Parameters:

dividend	Dividend of the division
divisor	Divisor of the division

Returns:

Remainder of the division

1.36. double sq_root (double *value*)

Get the square root of a number.

Parameters:

value	Radicand

Returns:

Result

1.37. void transpose(double* *matrixIn*, double* *matrixTranspose*, double *rows*, double *columns*)

Transpose a matrix.

Parameters:

matrixIn	Matrix whose transpose we want to obtain
matrixTranspose	Matrix where the result is saved
rows	Number of rows of *matrixIn*
columns	Number of columns of *matrixIn*

1.38. void vector_minus_value (double* *vectorIn*, double *value*, double* *vectorMinus*, int *positions*)

Subtract the same value from each position of a vector.

Parameters:

vectorIn	Each position is the minuend of each subtraction
value	Common subtrahend of each subtraction
vectorMinus	Vector where the result is saved
positions	Size of *vectorIn*

1.39. void vector_minus_vector (double* *vector1*, double* *vector2*, double* *vectorResult*, int *positions*)

Subtract one vector from another.

Parameters:

vector1	Minuend vector
vector2	Subtrahend vector
vectorResult	Vector where the result is saved
positions	Size of *vector1* and *vector2* (they must have the same number of elements)

1.40. double vector_mult (double* *vector1*, double* *vector2*, int *positions*)

Multiply two vectors.

Parameters:

vector1	First factor
vector2	Second factor
positions	Number of elements of each vector

Returns:

Product

1.41. void vector_mult_matrix (double* *vector*, double* *matrix*, double* *vectorResult*, double *positions_vector*, double *rows1*, double *columns1*)

Multiply a matrix by a vector.

Parameters:

vector	First factor
matrix	Second factor
vectorResult	Vector where the result is saved
positions_vector	Number of elements of *vector*
rows1	Number of rows of *matrix*
columns1	Number of columns of *matrix*

1.42. void vector_scale (double* *vectorIn*, double* *vectorOut*, double *value*, int *positions*)

Divide each element of a vector by the same value.

Parameters:

vectorIn	Each element of this vector is the dividend of each operation
vectorOut	Result of the division
value	Divider of each operation
positions	Number of elements of *vectorIn*

1.43. void write_double_binary(double* *matrixOut*, int *rows*, int *columns*, char* *fileName*)

Write an image into a binary file (row by row).

This function must be used to write images implemented with double type of data.

Parameters:

matrixOut	Image to save in the binary file
rows	Number of rows of *matrixOut*
columns	Number of columns of *matrixOut*
fileName	Name of the generated file

1.44. void write_short_binary(short int* *matrixOut*, int *rows*, int *columns*, char* *fileName*)

Write an image into a binary file (row by row).

This function must be used to write images implemented with short int type of data.

Parameters:

matrixOut	Image to save in the binary file
rows	Number of rows of *matrixOut*
columns	Number of columns of *matrixOut*
fileName	Name of the generated file

1.45. void write_txt(double* *matrixOut*, int *rows*, int *columns*, char* *fileName*)

Save a matrix into a .txt file (row by row).

This file is generated in the bin folder of the project

Parameters:

matrixOut	Matrix to be saved in the .txt file
rows	Number of rows of *matrixOut*
columns	Number of columns of *matrixOut*
fileName	Name of the output .txt file. The extension must also be written in this parameter

ANEXO 3. ORGANIZACIÓN DE DIRECTORIOS

En el presente anexo se realizará una breve explicación de la estructura general de los directorios utilizados durante el desarrollo del Proyecto Fin de Grado y, posteriormente, se explicará el contenido y la organización de cada uno de ellos.

En primer lugar, en la figura A.3.1 podemos observar la estructura de directorios utilizada:

Fig. A.3.1. Estructura de directorios utilizada

El contenido de estos directorios es el siguiente:

- *CMakeLists_Luna_Kepler*: Ficheros de configuración de dependencias de librerías de los proyectos desarrollados. Debido a que durante la realización del Proyecto Fin de Grado se lanzó una nueva versión del entorno de desarrollo *Eclipse,* se proporciona un fichero de configuración para cada versión del mismo.
- *doc*: Memoria y anexos desarrollados.
- *HAnDy_library*: Ficheros fuente necesarios para la implementación de la librería desarrollada en este Proyecto Fin de Grado.
- *projects*[23]: Los proyectos desarrollados para componer la cadena de procesado de imágenes hiperespectrales.
- *scripts*: Scripts de configuración, compilación y ejecución de cada uno de los proyectos aportados en el directorio *projects*.

A continuación, aportamos una breve explicación del contenido de cada uno de los directorios mencionados anteriormente.

A.3.1. CMakeLists_Luna_Kepler

En la figura A.3.2 podemos observar los dos ficheros que componen este directorio.

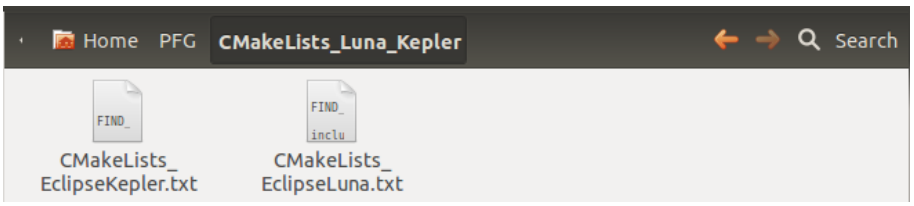


Fig. A.3.2. Contenido del directorio CMakeLists_Luna_Kepler

El primero de ellos es la configuración del programa en el caso de que se utilice Eclipse Kepler; por su parte, el segundo es el equivalente para el caso de utilizar Eclipse Luna.

[23] Todos los proyectos que se proporcionan están desarrollados en Eclipse Luna; si su versión es anterior, deberá regenerar el código Orcc antes de ejecutar el *script* correspondiente.

A.3.2. doc

El contenido de este directorio se muestra en la figura A.3.3, en ella se observa que, en la carpeta *doc*, se recogen tanto la memoria como sus respectivos anexos en formato *.pdf*.

Fig. A.3.3. Contenido del directorio doc

A.3.3. HAnDy_library

En este directorio se encuentran todos los archivos que componen la librería desarrollada a lo largo del Proyecto Fin de Grado. Los ficheros que la componen son los que aparecen en la figura A.3.4.

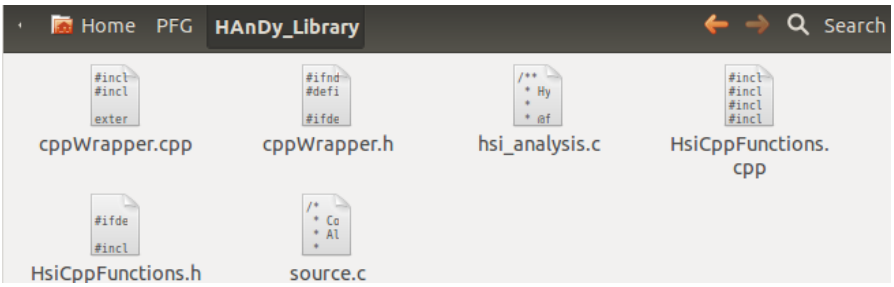


Fig. A.3.4. Contenido del directorio HAnDy_library

Como podemos observar, en la primera versión de esta librería se ha necesitado desarrollar código en lenguaje C y, a su vez, código en lenguaje C++, por lo que ha sido indispensable crear un *wrapper* entre ambos lenguajes para poder hacer compatible su utilización. La razón por la que se utiliza lenguaje C++ es que existen funciones de alta complejidad matemática pertenecientes a las librerías que se instalaron y configuraron en el Anexo 1 (ITK,OTB y VXL), que están implementadas en este lenguaje.

Una vez explicado el por qué de la utilización de dichos ficheros, aportamos una breve explicación del contenido de cada uno de los archivos presentes en este directorio:

- *source.c*: Este archivo contiene las funciones relacionadas con la lectura de ficheros binarios que contienen la información de la imagen hiperespectral. Es necesario para el desarrollo de la interfaz de entrada del sistema.
- *hsi_analysis.c*: En este fichero se encuentran las funciones básicas (en lenguaje C) y las llamadas a funciones complejas (en lenguaje C++) de la librería desarrollada. Este fichero contiene el grueso de las funciones desarrolladas en esta librería.
- *cppWrapper.h*: En este fichero se recogen los prototipos de las *funciones enlace*[24] entre lenguaje C y lenguaje C++.
- *cppWrapper.cpp*: Este fichero sirve de enlace entre ambos lenguajes. Contiene las *funciones enlace* y, a su vez, realiza la llamada a la función en lenguaje C++ correspondiente.
- *HsiCppFunctions.h*: En este fichero se recogen los prototipos de las funciones implementadas en C++.
- *HsiCppFunctions.cpp*: Este fichero incluye todas las funciones desarrolladas en C++ necesarias para el correcto funcionamiento del sistema.

[24] *Función enlace*: Mecanismo que posibilita la llamada a métodos C++ desde funciones C.

A.3.4. Projects

Este directorio contiene los proyectos que se han desarrollado para la comprobación del correcto funcionamiento de la librería. En la figura A.3.5 podemos observar los tres proyectos finales realizados. También se puede ver que aparece un cuarto directorio denominado *RemoteSystemsTempFiles*; esto se debe a que este directorio también es utilizado como *workspace* del entorno de desarrollo Eclipse, y es creado de forma autónoma por dicha herramienta.

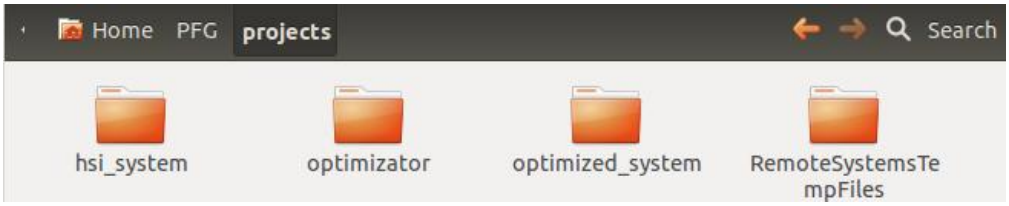


Fig. A.3.5. Contenido del directorio projects

La explicación de los tres proyectos realizados es la siguiente:

- *hsi_system*: En este proyecto se encuentra la cadena completa de análisis de una imagen hiperespectral. Esta cadena se encuentra dividida en actores, de tal forma que en cada uno de ellos se implementa una etapa de la cadena (PCA, VCA y LSU). Asimismo, también se aportan un actor *Source* que permite leer un fichero binario y un actor *Display* que genera la salida - también en un fichero binario -, constituyendo, por tanto, las interfaces de entrada y salida de la cadena de análisis.
- *optimizador*: Este proyecto contiene la etapa opcional (estimación del número de *endmembers* óptimo - *HySime*) generando a su salida un fichero denominado *Data.cal* que funciona como archivo de especificaciones (configuración de bandas y *endmembers* deseados) en el proyecto *hsi_system*.
- *optimized_system*[25]: En este fichero se aúnan todas las fases del análisis de la imagen hiperespectral, uniendo la fase de estimación de *endmembers* con las fases propias del análisis de una imagen. En este proyecto, el análisis de la imagen se hará siempre con el número óptimo estimado de bandas y *endmembers* de una imagen.

Tras la explicación de los proyectos realizados, pasamos ahora a estudiar la organización interna de los mismos. Para ello, explicaremos únicamente la estructura de uno de ellos - por ejemplo, del proyecto *hsi_system*-, puesto que dicha estructura es común a todos los proyectos. En primer lugar, la estructura general de cualquier proyecto es la que aparece en la figura A.3.6.

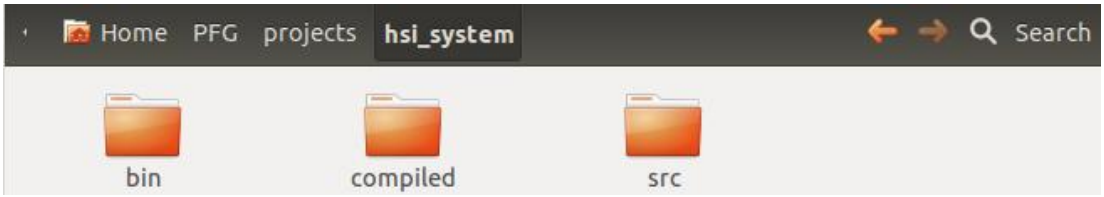


Fig. A.3.6. Contenido del directorio hsi_system

Como podemos observar, la estructura general se divide en tres directorios:

- *bin*: Este directorio contiene archivos de extensión *.ir* necesarios para la generación automática de código. Estos archivos son autogenerados por el entorno Eclipse.
- *src*: En este directorio se encuentran los archivos con extensión *.cal* que se utilizan en Orcc para implementar actores y definir *networks*.
- *compiled*: Este es el directorio de salida del código generado por Orcc y que, por lo tanto, tendrá el programa en lenguaje C y deberá incluir la librería generada.

[25] En este proyecto se modifica el código autogenerado por Orcc, por lo que cualquier cambio efectuado en el código proporcionado será ignorado.

El contenido de este último directorio se puede observar en la figura A.3.7.

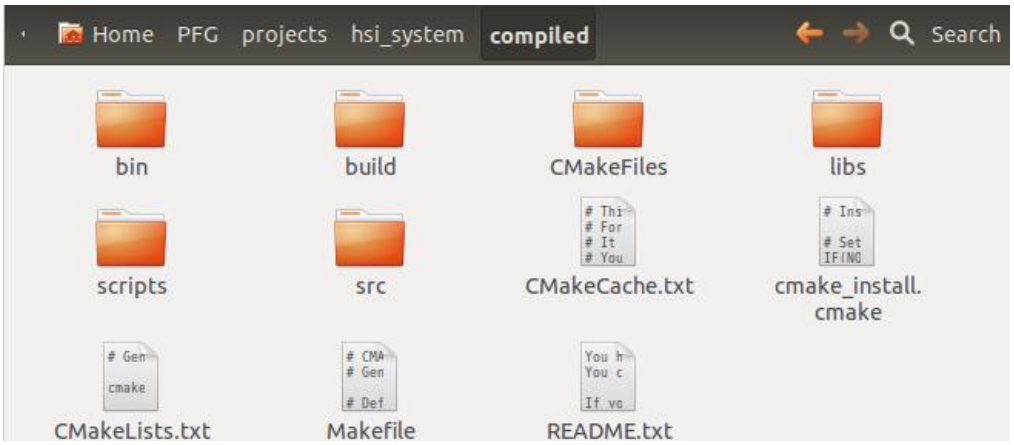


Fig. A.3.7. Contenido del directorio compiled

De los directorios que incluye la carpeta *compiled*, los que son de interés para el usuario son lo siguientes:

– *bin*: En este directorio se genera el ejecutable del programa realizado y, además, se almacena la entrada y se genera la salida de los distintos programas, tal y como podemos observar en la figura A.3.8.

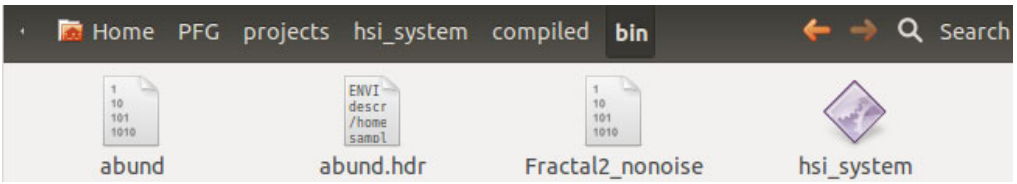


Fig. A.3.8. Contenido del directorio bin *de un proyecto*

– *src*: Como se puede observar en la figura A.3.9, en este directorio se encuentra el código autogenerado por Orcc en lenguaje C, así como un fichero de configuración en formato *.xcf* para organizar la distribución del programa entre varios procesadores.

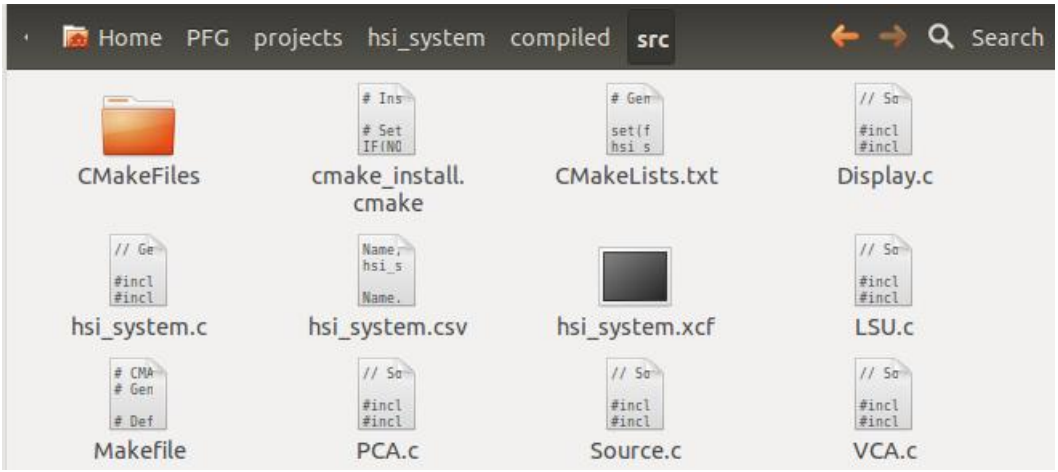


Fig. A.3.9. Contenido del directorio src *de un proyecto*

– *libs*: En este directorio se encuentran todas las librerías necesarias para que el programa se ejecute de manera satisfactoria. Esto implica que la librería desarrollada debe ser incluida en este directorio si queremos que funcione correctamente. En concreto, dicha librería está fomada por funciones nativas, por lo que, para seguir con la organización de Orcc, habría que incluirla -junto al *CMakeLists.txt* de configuración- en el directorio *orcc-native* que se observa en la figura A.3.10.

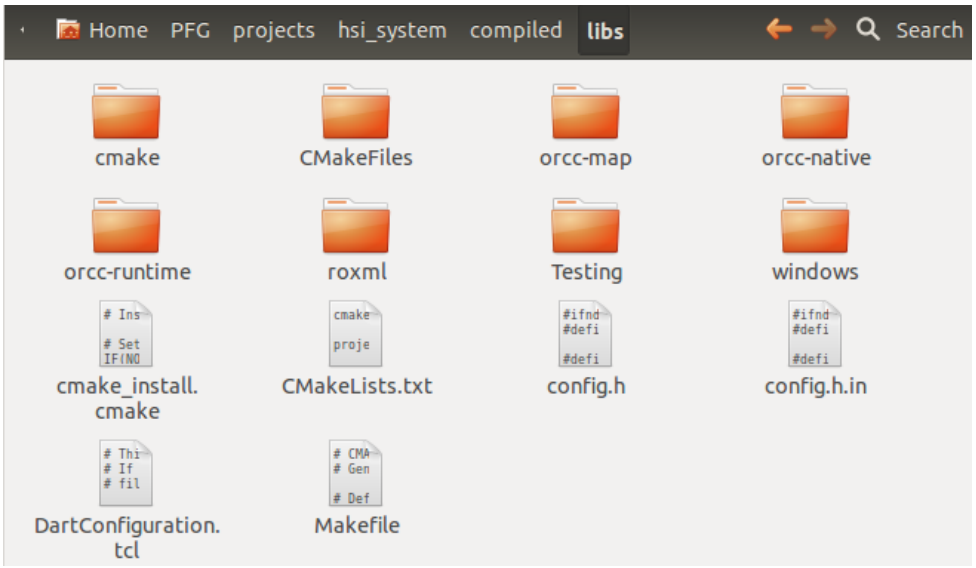


Fig. A.3.10. Contenido del directorio libs *de un proyecto*

A.3.5. Scripts

En este último directorio se aportan los *scripts* de configuración, compilación y ejecución de cada uno de los proyectos. Estos ficheros incluyen la copia automática de todos los ficheros de la librería en sus directorios correspondientes, la compilación, también automática, y la ejecución con los parámetros concretos propios de cada programa. Dicho directorio está organizado, como se puede observar en la figura A.3.11, por proyectos.

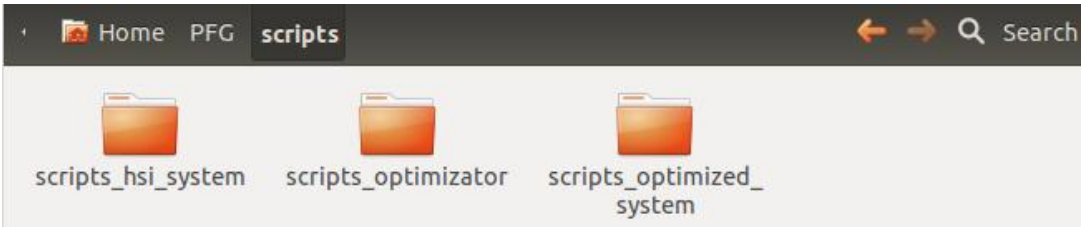


Fig. A.3.11. Contenido del directorio scripts

En cada uno de los directorios se aporta un *script* propio para cada versión de Eclipse, tal y como observamos en la figura A.3.12.

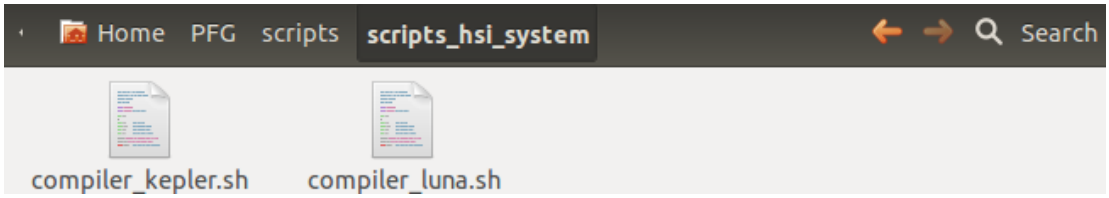


Fig. A.3.12. Contenido del directorio scripts_hsi_system